MacTech Magazine
Vol. 20, No. 03 • March 2004

MacTech®

The Journal of Macintosh Technology and Development

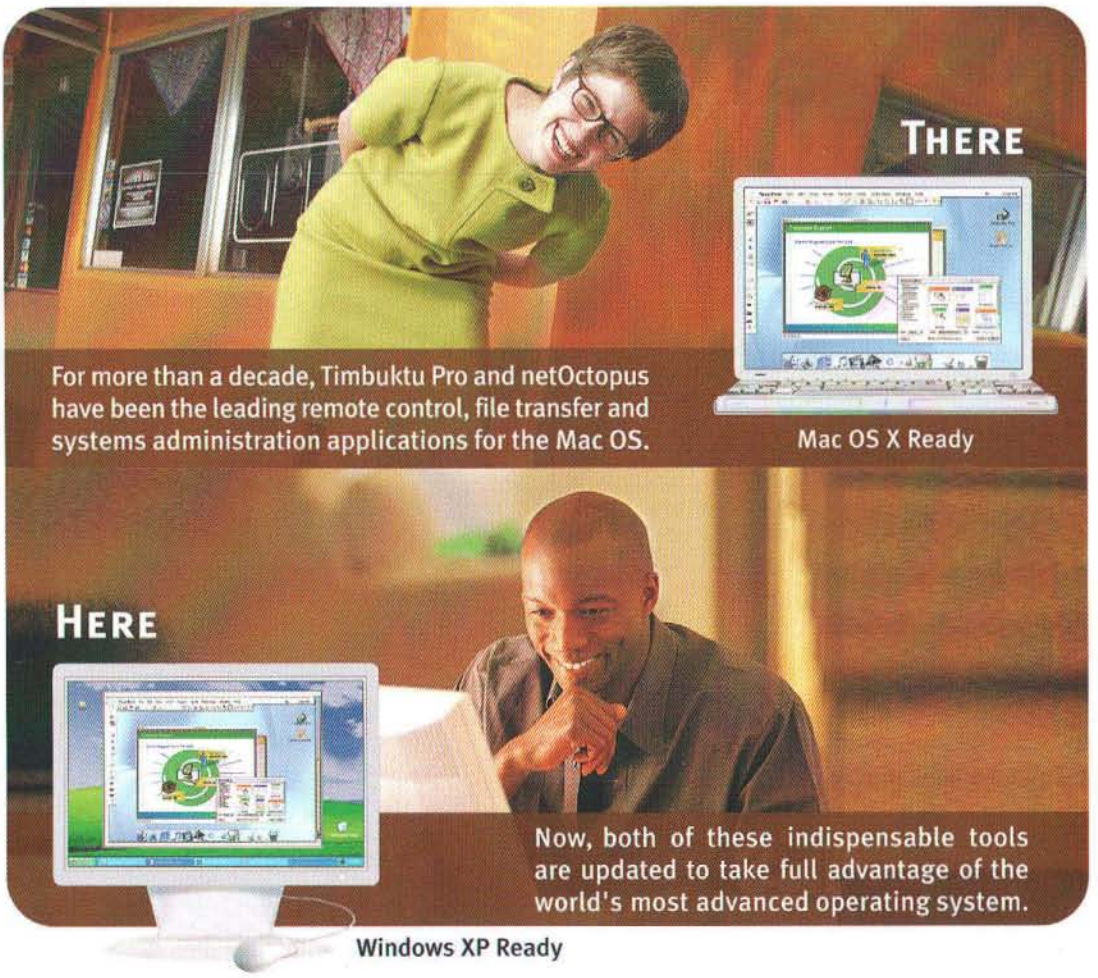
The Cool Side of the Moon

The Eclipse IDE on the Macintosh

by Ruben Kleiman

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Stay In Control Wherever You Go.



Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.

http://www.timbuktupro.com

netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.

http://www.netoctopus.com

Learn more, try it, or buy it online. Call us at 1-800-485-5741.

timbuktu® • netOctopus®

netopia.

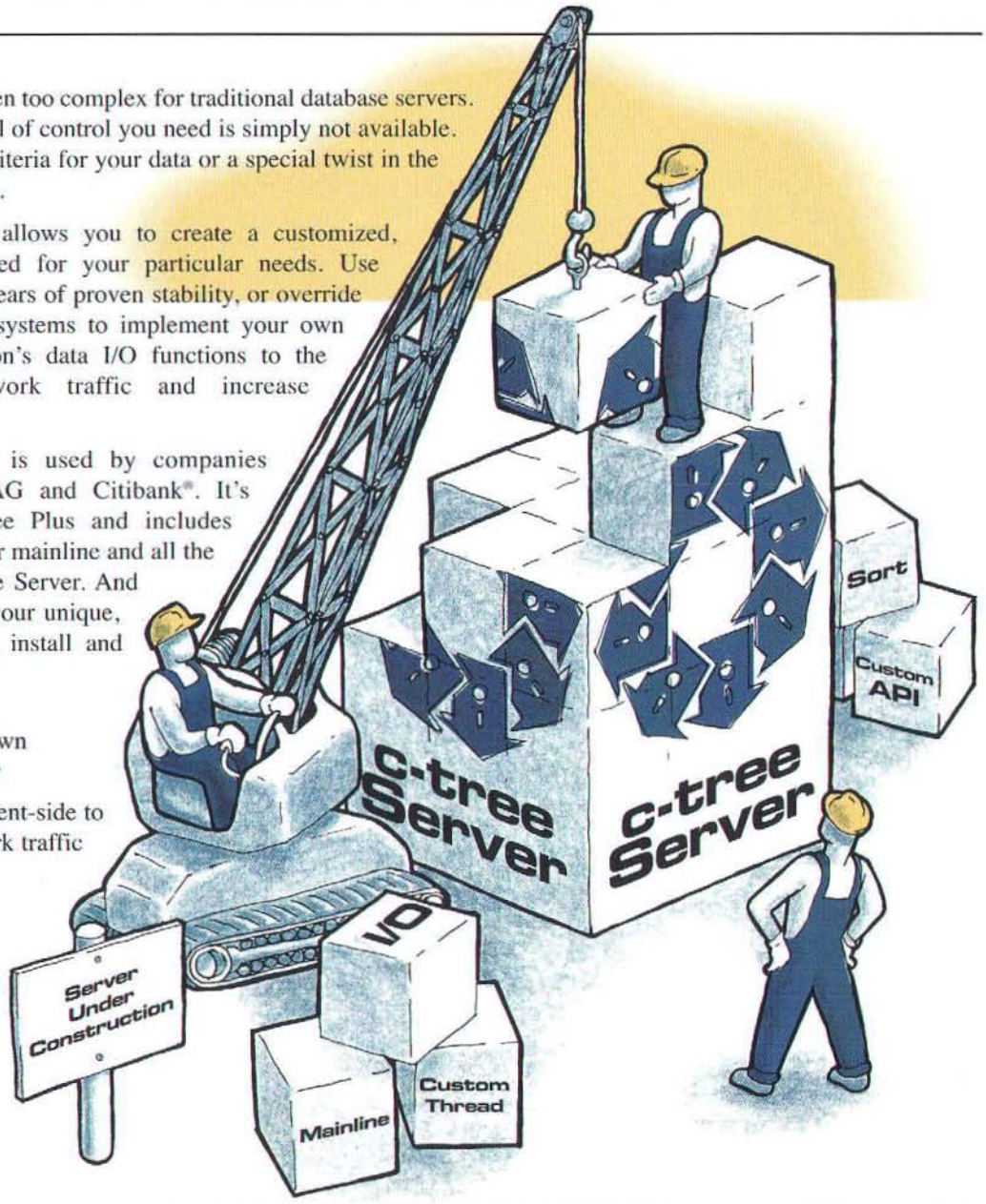
CUSTOMIZE
YOUR DATABASE SERVER WITH THE C-TREE® SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree® Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank®. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!

- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing



Visit www.faircom.com/ep/mt/sdk today to take control of your server!

FairCom®
www.faircom.com

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • info@faircom.com

MacTech®
The Journal of Macintosh Technology & Development

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

March 2004 • Volume 20, Issue 3

Getting Started with Cocoa...

As anyone who knows me well can tell you, I *love* programming the Mac. And I'll play with just about anything out there that lets me build a Mac app or somehow control the Mac interface. When I first installed Mac OS X on my laptop, I immediately downloaded the developer tools, dug into the doc and started building Cocoa applications. And immediately got stuck.

Like a lot of other folks out there, I was a bit overwhelmed with all I needed to learn to truly understand the mechanics of Cocoa. For example, I know that I can use Interface Builder to lay out the user interface for an application. But why do I need to control-click-drag from one interface element to another? What are outlets? Actions? Targets? What the heck is a File's Owner or a First Responder?

These questions are the tip of the iceberg, but common to everyone when they first take on Cocoa. There are a number of Cocoa books out there, some of them quite good, but they all make some pretty big leaps in logic – not a big problem for professional developers, but those leaps can leave beginners scratching their heads, trying to decide if Cocoa is really hard to master or if they are just not up to the task. Let me assure you – Cocoa really *is* a challenge, even for the experienced developer.

Over the next few months, we're going to dig into Cocoa and get the basics down. Build up a deep enough understanding of Cocoa mechanics so that all those Cocoa books on your shelf make a lot more sense.

Big Nerd Ranch

Before we start, I wanted to digress for just a moment, talk about a great experience I just had. A few weeks ago, Scott Knaster and I were fortunate enough to head down to historic Banning Mills, about an hour southwest of Atlanta for a week of intensive Cocoa training with the folks at *Big Nerd Ranch*.

> *Banning Mills* is located on about 700 acres along the picturesque Snake Creek in Carroll County, Georgia. Originally home to the Creek and Cherokee nations, in the mid-1800s it became home to one of the earliest textile mills in the south. Over time, a masonry mill was built, the mill town grew, wool and cotton yarn was produced and processed into cloth. Paper and lumber products, meal, and flour were also produced. The area become an important part of the southern economy.
> Towards the end of the Civil War, General William Tecumseh Sherman was sent to find the mills and destroy them. Because of the unique geography of the area (the mills are in an extremely well-hidden gorge), he was unable to locate the mills and they survived.
> Check out http://historicbanningmills.com

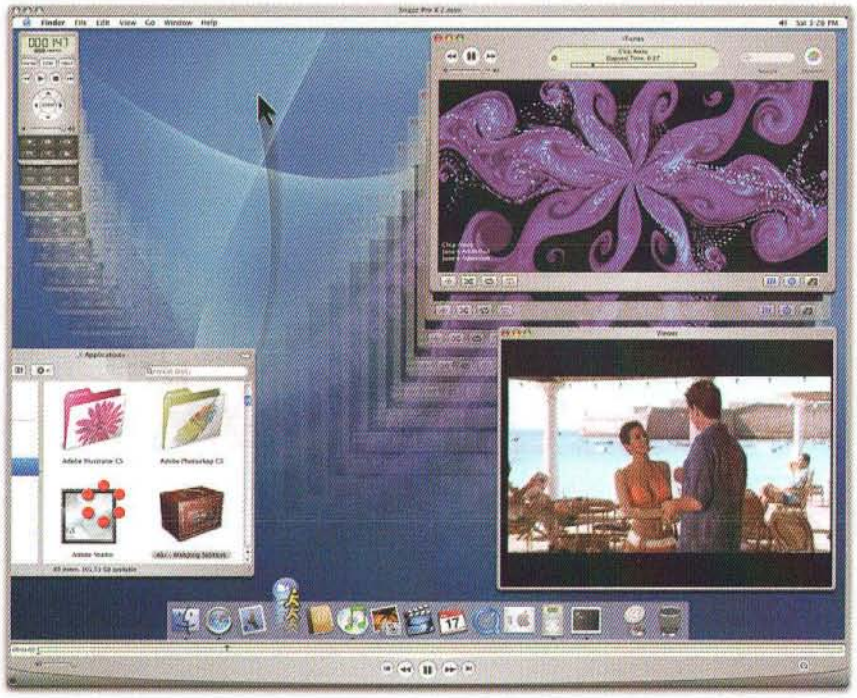
I arrived at the ranch on Sunday evening, with the class scheduled to start Monday morning. Checked into my cabin – rustic with a standalone gas fireplace that really heated up the room – not so rustic with jacuzzi tub, phone, and TV with cable as well. Call it "charming with modern amenities". Once I unpacked, I headed over to the main lodge. To my surprise and delight, there was a nice spread of food laid out, complete with a drinks cooler (with all my favorites) and an ice cream freezer with about every kind of ice cream you could imagine. And, best of all, everyone in the class drifted in and we all sat around the dining table eating and getting to know one another. ScottK kept us all entertained with war stories from his days at Apple, General Magic, and Microsoft. Somehow, each story ended up with Scott making friends with a newly minted billionaire. Sadly, none of them was me.

The classes themselves were held in an upstairs meeting room at the lodge. Rustic, yet comfortable. A door off of the classroom led onto a large deck with a majestic view of Snake Creek and the surrounding rocky terrain. At night (and every night was a *very* late night) we'd go out onto the deck, talk Cocoa, and take in a sky full of stars. No light pollution at Big Nerd Ranch.

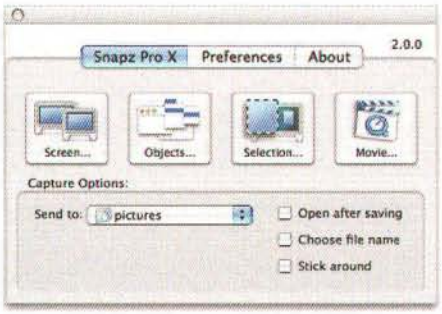
Our teacher, Aaron Hillegass, was exceptional. The class was based on an upcoming second edition of his book, *Cocoa Programming for Mac OS X*. The book was presented in loose-leaf format. The downside of working from a work-in-progress is that the code on disk frequently differed from the code on the pages. Sometimes the book was right,

Dave Mark is a long-time Mac developer and author and has written a number of books on Macintosh development, including *Learn C on the Macintosh, Learn C++ on the Macintosh,* and *The Macintosh Programming Primer* series. Interested in Learning Cocoa? Be sure to check out Dave's web site at http://www.spiderworks.com.

If a picture is worth a thousand words, imagine how priceless a movie would be...



Snapz Pro X 2.0 allows you to effortlessly record anything on your screen, saving it as a QuickTime® movie that can be emailed, put up on the web, or passed around however you please.



Why take a static screenshot when Snapz Pro X 2.0 makes creating a movie just as easy? Snapz Pro X 2.0 does that, and so much more -- what a difference a version makes! Download a free demo version from our web site today and see for yourself:

http://www.AmbrosiaSW.com/

AMBROSIA® SOFTWARE INC

Snapz Pro X 2.0

AMBROSIA®
SOFTWARE INC

Snapz Pro X

Figure 1. *The graduating class, from left to right: Rick Steele, Bryan Lockwood, Chris Campbell, Scott Knaster, Aaron Hillegass, Dave Mark, Allan Hoeltje, Jorge Ramirez, Mark Dalrymple, and Derek Chan. Aaron was the head cheese, and Chris and Mark his able assistants.*

sometimes the disk was right. I typed in all the code and fixed any anomalies as I went along. Aaron was terrific about answering questions and helping debug our apps. I liked the fact that I typed in the code, as I was sure that I understood what ultimately ended up working.

As far as content goes, we covered Objective-C in detail, spent some time on Xcode, Interface Builder and the Debugger, then got into Cocoa itself. Rather than go over the hierarchy class-by-class, Aaron presented overarching concepts, then introduced the classes in place as they occurred in the code. For example, he started by introducing a basic user interface with buttons and a text field, then introduced classes like NSControl (and subclasses NSButton, NSSlider, NSTextField) and NSAppController. Rather than tour through each class, Aaron started in Interface Builder and wired the app together, giving an overview of each class as it was hooked in place. This gave us less of an exhaustive view of the class hierarchy and much more a sense of the Cocoa design philosophy. An excellent approach.

We explored Helper Objects, Bindings and NSController (and the very important Key-value Coding concept), NSUndoManager (holy cow - it really is easy to add undo to your app!), Archiving and the Document Architecture, Nib files and NSWindowController, User Defaults, Notiffications, Alert Panels, Localization, Custom Views, Events, Responders, Fonts, Pasteboards, Categories, Drag-and-Drop, NSTimer, Sheets, NSFormatters, Printing, Menus, NSTextView, building your own Interface Builder palettes, AppleScript, and OpenGL.

An amazing amount of material, crammed into way too little time. Yet somehow it all worked. I came away remembering much of what was taught, but more importantly, I had enough resource material to recreate anything I'd forgotten.

If you can afford it, I would strongly recommend *Big Nerd Ranch*. Come prepared, though, as the pace is intense. At the very least, before you head out, I would read the document *ObjC.pdf* in this directory:

```
/Developer/Documentation/Cocoa/Conceptual/ObjectiveC/
```

Do your homework first, and I know you will find the experience to be tremendously rewarding. Check out the Big Nerd Ranch website at http://www.bignerdranch.com. You can call them at (678) 595-6773.

IN THE SOPHOS ZONE, VIRUSES NEVER INTERRUPT YOUR WORKFLOW

Sophos Anti-Virus provides total protection against all known viruses on Mac OS X. Its GUI enables users to perform an immediate scan of any accessible file, folder and volume, while InterCheck technology keeps users safe by intercepting all file accesses and scanning for viruses in the background in real-time. In addition, every license includes 24x7x365 technical support. In the Sophos Zone, viruses don't stand a chance.

For a free, fully supported evaluation, visit: www.sophos.com/av_mac

www.sophos.com/av_mac
Tel 888-216-6703

SOPHOS
engineered for business

The Number Generator

Perhaps the best way to get started with Cocoa is with a simple application that makes use of both Interface Builder and Xcode, just so you can get a feel for the development process. Don't worry about the concepts. Just click and drag and follow along. Over time, the steps you take in this program will become second nature.

This is the very first project in the upcoming second edition of Aaron Hillegass' *Cocoa Programming for Mac OS X*. The book should hit the streets in a few months.

The program, called *RandomApp*, puts up a window with a couple of buttons and a text field. One button initializes the system's random number generator and the second button asks the generator to generate a random number between 1 and 100. The number is displayed in the text field. The result is shown in **Figure 2.**

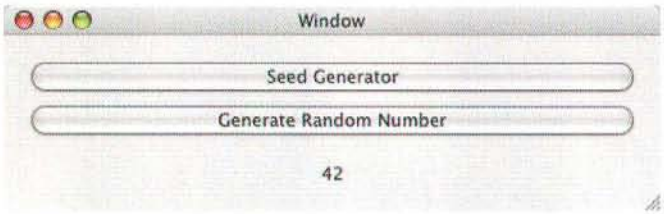


Figure 2. *RandomApp in action.*

In this month's installment, we'll create a project in Xcode, then use Interface Builder to build and test the interface. Next month, we'll add the wiring to tie the buttons to the appropriate code, then add the code itself. Sounds good? Let's get started...

Create a New Xcode Project

Launch Xcode and select *New Project...* from the *File* menu. When prompted, select *Cocoa Application* as the project type (see **Figure 3**). *Cocoa Application* is under the *Application* section. Be sure you do *not* select *Cocoa Document-based Application*. Click the *Next* button.

Be sure you have the latest and greatest version of the developer tools installed. I wrote this column using Xcode version 1.1. You can download the tools at http://developer.apple.com. You'll need to join *Apple Developer Connection* but it is free to join.

Figure 3. *Build a new Cocoa Application project.*

Next, you'll be prompted to enter a project name and project directory (see **Figure 4**). Name the project *RandomApp* and save it in an appropriate directory. Click the *Finish* button.

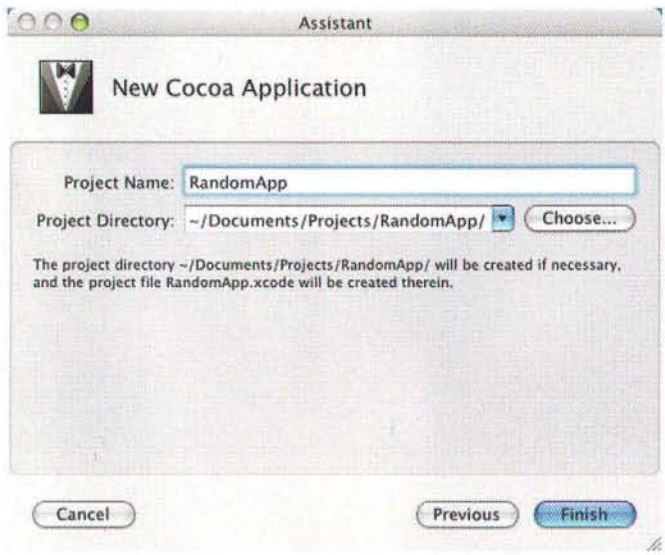


Figure 4. *Name the project RandomApp and save it in an appropriate directory.*

Your project window will appear and Xcode will immediately begin indexing the project files. This happens in the background and is the reason Xcode's searches are so incredibly quick.

The left pane of the project window lists the *Groups & Files* that make up the project. One of these groups is named *NIB Files*. Open the *NIB Files* triangle and you will find the nib file for this project, named *MainMenu.nib* (see **Figure 5**). Double-click on *MainMenu.nib* to open the nib file in Interface Builder.

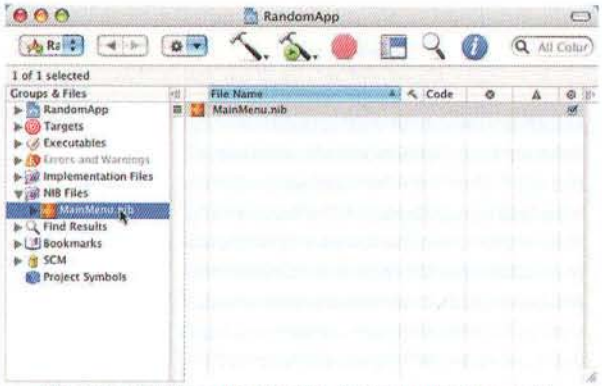


Figure 5. *The RandomApp project window, with MainMenu.nib selected.*

Laying Out the User Interface

Interface Builder is an incredibly powerful tool. It not only allows you to design your application's user interface, it also let's you define the interactions between the objects that make up your application. Mastering the intricacies of connecting objects will take time. We will definitely spend a lot of time experimenting over the months ahead. For now, just follow along and get RandomApp up and running. You'll get a basic sense of the interaction between Interface Builder and Xcode, and you'll feel a bit more comfortable knowing all your tools are in place and working properly.

When Interface Builder appears, you should see a menu window labeled *MainMenu.nib – MainMenu*, a window labeled *Window*, a main window labeled *MainMenu.nib*, and a floating palette whose name changes depending on the icon selected from the row at the top of the palette. In the palette window, click on the icon with a button and a slider (2nd from the left, as shown in **Figure 6**) to bring up the controls palette.

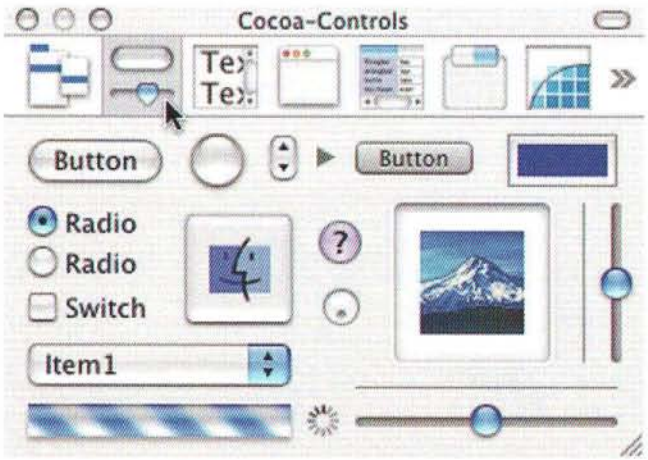


Figure 6. *Interface Builder's controls palette*

Click on the button in the upper-left corner and drag it onto the window labeled Window. Take a look at **Figure 7**.

Notice that as the button neared *Window's* upper-left corner, A pair of blue dashed lines appeared. These guide lines tell you the item you are dragging is an appropriate distance from the nearest other interface element (in this case, the top and the side of the window). This is quite a handy way of lining up your interface objects.

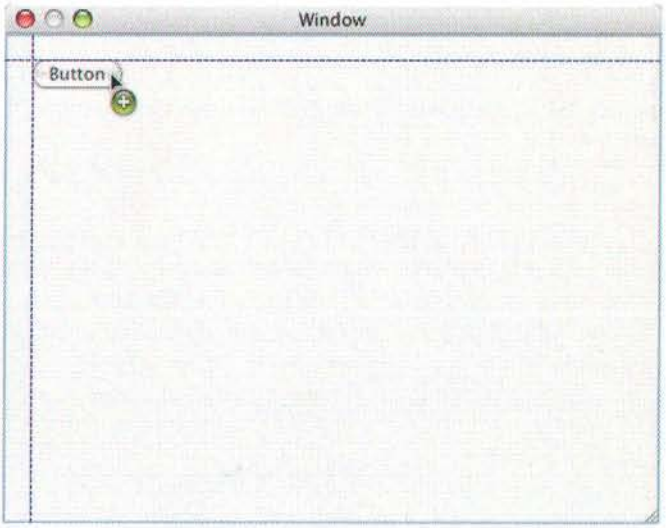


Figure 7. *Dragging a button from the palette onto the window.*

Next, click to select the button, then click on the right resize bits and drag the rectangle to the right, till the blue dashed line towards the right end of the window appear. Now you should have a button that is almost as wide as the window.

Double-click on the word *Button* and type *Seed Generator*, followed by a return. The button text should change appropriately.

Now drag out a second button underneath the first one. This time, there will be blue guidelines above and to the left of the button as you drag it out. These guides will keep the button the proper distance from the top button and the left side of the window.

Drag the right side of the button to make it as wide as the top button. Double-click on the word button and change its text to say *Generate Random Number*.

Figure 8 shows what we have so far...

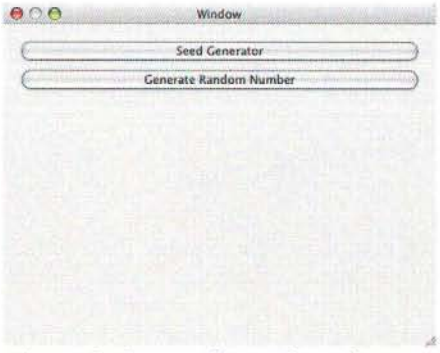


Figure 8. *Our window with two buttons.*

Our next step is to drag out a static text field onto the window. In the palette window, click on the 3rd icon in the top row (the one that looks like a scrolling text field). The palette title should change to read *Cocoa – Text* and a series of text related objects should appear in the palette window.

Click and drag a static text item from the lower right corner of the palette onto the window we are building. Notice that you have a choice of several different size static text items. The one we want is the largest one and is labeled *System Font Text*. Just as you did with the buttons, drag the text to the left side of the window, underneath the 2nd button. Use the blue guide lines to make sure your spacing is right.

Click and drag on the right side of the text field to resize it to the same width as the buttons.

Next, select *Show Info* from the *Tools* menu to bring up a floating inspector window. When the *Info* window appears, click on the static text field you just created. The fields of the *Info* window will change to reflect the settings of the static text field. Change the *Title* field to the number *0* and click on the centering icon in the *Alignment:* field. **Figure 9** shows my settings.

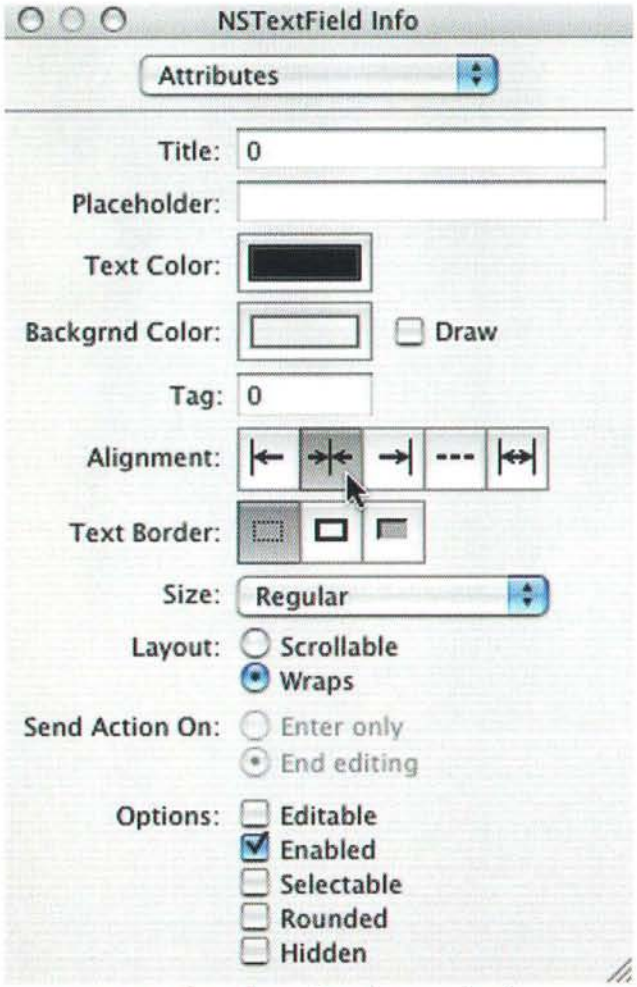


Figure 9. *The info window, showing the changes we've made to the NSTextField.*

Finally, click and drag the window to make it shorter. Keep it the same width, though. **Figure 10** shows my final window.

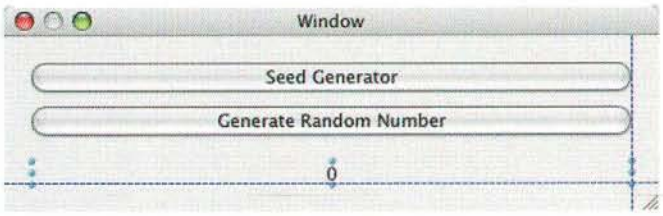


Figure 10. *The final window.*

Test Drive the Interface

Now that your interface is finished, let's take it for a spin. Select *Test Interface* from the *File* menu. The RandomApp window should appear, with the title *Window* and the buttons and text as we've laid them out.

The buttons should be operable, though they won't actually do anything. When you are done playing, select *Quit NewApplication* from the *Interface Builder* menu. This will quit the test drive, not Interface Builder itself.

Till Next Month...

In next month's column, we'll use Interface Builder to create a new class specifically designed to seed the random number generator and generate a new random number. We'll create an instance of the class and wire the two buttons to the instance's methods that seed the generator and generate a new number. We'll then go into Xcode and add the code to make the application complete.

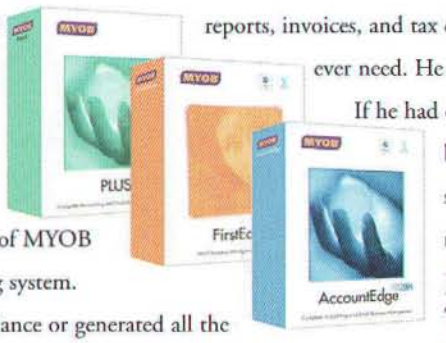
If you dare, take some time to play with Interface Builder and customize your interface. Change the window's title from *Window* to *RandomApp*. Change the menus. And if you double-dare, try your hand at finishing the program yourself.

Before you start mucking, thought, please save your changes and make a copy of your project directory. Next month, we'll want to start exactly where we left off. See you then! ☺

See Dick. See Dick run his business with software that wasn't written and designed for his Macintosh. Poor Dick.

A moment of silence for Dick, please. A good guy with a good small business, but his accounting software was one of those PC transcription jobs, not pure MAC like MYOB AccountEdge and MYOB FirstEdge.

If only he'd known about the amazing capacity of MYOB software to bring out the best in his MAC operating system. He could have tracked and managed finances at a glance or generated all the reports, invoices, and tax documents that he and his accountant would ever need. He could have spent more time with his clients.



If he had only known that MYOB develops the world's best selling MAC small business management software for lots of good reasons, this story might have had a happy ending. Sorry Dick.

MYOB,
THE MAC ANSWER.

By Scott Knaster

Your Attention, Please: Alerts and Sheets in Cocoa

I have seen the light, and it turns out to be cocoa-colored. As you already know if you have read Dave Mark's Getting Started column, he and I spent a transformational week taking the Cocoa Bootcamp class at the Big Nerd Ranch. During this week, the other students and I were thrown into an ocean of Cocoa, but we didn't drown. Class instructor Aaron Hillegass' strategy is to take attendees through an enormous volume of material, and it works: you don't remember everything about everything, but a surprising amount sticks to your brain. The result is that the class is like a Cocoa Magic Decoder Ring. Now, I can look at Cocoa books, documentation, and code, and figure out what's going on – before, I couldn't.

As Dave mentions in his column, the class is held in rural Georgia, which is probably not the first place you think of when looking for a hotbed of Cocoa development. But upon arriving at the lodge, I figured out why this was such a good place for Cocoa training: one of the region's legendary characters is Chief William McIntosh. His face is all over the lodge, and figure 1 shows you what he looked like. McIntosh, Cocoa – get it?

Figure 1. *Chief William McIntosh is an appropriate historical figure in the area where the Big Nerd Ranch holds its classes.*

In this month's column, I'm going to start by talking a little about some observations and "aha" moments that happen as you're trying to figure out what Cocoa is all about. Then, I'll go through a gentle example of how you get something done in Cocoa: putting up an alert.

Drink the Cocoa

If you're an old-school programmer with experience writing Macintosh or Windows apps, several obstacles pop up immediately when you try to learn Cocoa. The roadblocks include:

- Learning to read and think in Objective-C.
- Learning to use and trust frameworks in general.
- Figuring out how to use the Cocoa frameworks specifically.

Learning Objective-C is the first challenge to working in Cocoa. Although you can use Java with Cocoa, everything is designed first with Objective-C in mind. Objective-C and Cocoa together have some rules and conventions that will make you nuts at first, such as sending messages (which turn into method calls) using square brackets, formal parameter names (labels) in method calls, using the "NS" prefix for lots of symbols, including class names, constants, and plain functions, and no explicit "override" keyword . You get used to most of these, but they're a pain until you get used to them. There are also some important new concepts to learn when programming in Cocoa with Objective-C, such as *delegates* (customizing the way objects behave through a modern version of what used to be called *books* in the old Mac days), *categories* (a way to add methods and instance variables to a class), and *nib files* (the output of Interface Builder, consisting of encoded object instances that can be decoded and used by your running application).

If you've programmed with other object-oriented frameworks, you'll have an advantage in figuring out Cocoa. One of the trickiest parts is knowing which piece to use to solve a particular problem. The best way to gain this knowledge is through experience, which means it's hard when you're getting started.

Scott Knaster attended Cocoa Bootcamp at Big Nerd Ranch (http://www.bignerdranch.com/classes/cocoa1.shtml). He did not go Cocoa Loco but he liked it very much. Scott counts the days until pitchers and catchers report.

How can you get around these obstacles? First, bootstrap yourself into Cocoa with a good class or book (or both). If you like learning by doing, Cocoa Bootcamp at the Big Nerd Ranch is a fantastic experience. Or, you can read and work through the book **Cocoa Programming for Mac OS X**, which is basically the portable version of the class. For those who like to learn by reading before jumping in and trying things, check out **Cocoa in a Nutshell**. The two books together form a fine combination.

As soon as you're to the point where you can look at Cocoa code in Objective-C and have a clue about what's going on, make sure you retain and build your skills by writing Cocoa code as often as possible. Fresh knowledge tends to evaporate if it's not used.

Rich Cocoa

Cocoa provides an amazing box of goodies for programmers to play with. Cocoa is so vast that if you printed out all the Cocoa APIs and laid them out end to end, you would have to walk a very long way indeed to read it all. This month and in future editions, we'll explore cool things you can do in Mac OS X using Cocoa, and I'll try to make sure we have fun while we're digging around. We'll start out this month with a gentle example – putting up an alert – that will also serve to introduce a little of the Cocoa way of doing things.

Before we write code, lets review a bit of user interface business. According to Apple's Aqua Human Interface Guidelines, Aqua supports three kinds of dialogs: modeless, document modal (also called *sheets*), and application modal. In this month's column, we're going to work with modal dialogs, which are also called *alerts*. We'll discuss both document modal and application modal alerts. We'll talk about how the recommended way to create alerts has changed recently in Cocoa, and what to do about it.

Our example app is going to create an alert that lets the user confirm whether to revert to factory settings. What's being reverted, and the actual reverting itself, are left as exercises for you and me to do later. Our application will create windows that look like **Figure 2**. The window includes four buttons: two that put up application modal dialogs, and two for document modal dialogs (sheets). There are two for each because we're demonstrating two different techniques for putting up alerts: one that was introduced in Mac OS X 10.3, and one that works in previous versions as well as 10.3.

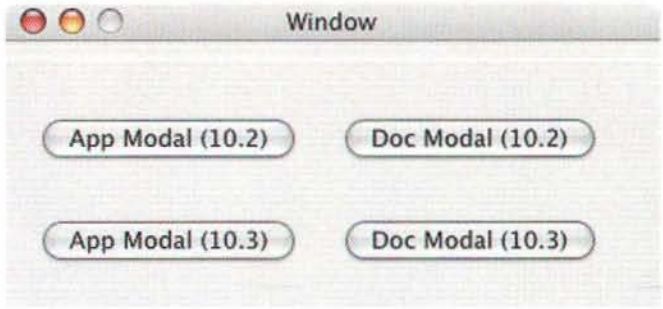


Figure 2. *This is what our document window will look like.*

It's Time to Start

Let's start by creating the project in Xcode. Choose New Project, then Cocoa Document-based Application. Mine is named RedAlert. As usual with Cocoa projects, we'll start with the fun stuff by laying out the user interface and object connections in Interface Builder (IB). We double-click MyDocument.nib and IB opens. First, we'll build the document window and the buttons that will appear inside them. Click over to the Cocoa Windows palette (that's the fourth palette over, signified by a tiny picture of a window in the toolbar), and drag out a window. Then, click the second tab in the toolbar to go to the Controls palette, and drag out four buttons to make your window look like the one in **Figure 2.**

Now we need a way to make the buttons work. Our big plan is to have the buttons send messages to a controller object when the user pushes them. We're going to make a new class called AppController, which will be a subclass of NSObject. Click Classes in the MainMenu.nib window, then click NSObject. Choose Classes ® Subclass NSObject to make the new class. (You can also right-click NSObject, or simply press return, to make a subclass.) Type a name for the new class: AppController.

NeXT, we have to get AppController objects ready to receive messages from the buttons. With AppController selected, choose Tools ® Show Info, and make sure Attributes is selected on the popup menu at the top. We have four different buttons that are going to send messages to objects of this class, so we'll define four actions that controller objects can perform. Click Add and type in the four actions, one by one: appModalNew, appModalOld, docModalNew, docModalOld. The info inspector should look like the one in **Figure 3.**

W WINDOWS | M MACINTOSH | X MAC OS X

Thinkfree

ThinkfreeOffice

COMPATIBLE WITH MICROSOFT WORD, EXCEL AND POWERPOINT

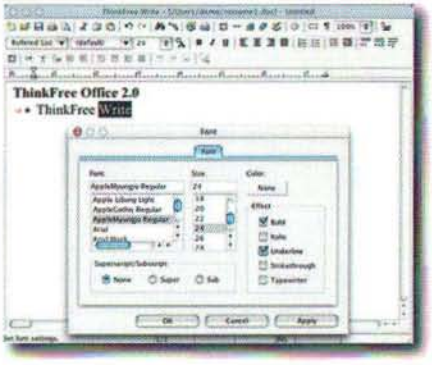
WORDPROCESSOR • SPREADSHEET • PRESENTATION GRAPHICS

The Affordable Office Alternative!

Three High-Performance Applications

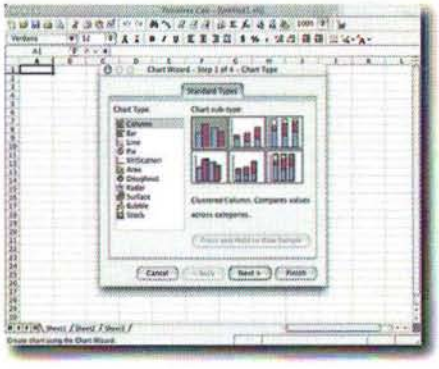
Thinkfree Write

Thinkfree Write is a powerful word processing application that enables you to create rich, professional quality documents and Web pages. You can insert tables, images, and clipart, or even apply custom layouts to your document...then effortlessly proofread your work with the easy-to-use spelling and auto-correction features.



Thinkfree Calc

Thinkfree Calc is a full-featured, easy-to-use spreadsheet application that can easily tackle the most complex analytical tasks with over 40 charts and 300 function capabilities. Thinkfree Calc opens, edits, and saves directly into the Microsoft Excel (.xls) format, so users can seamlessly share documents and collaborate with Microsoft Office users.



Thinkfree Show

Thinkfree Show enables you to create high-impact presentations including animation effects, drawings, images, clipart, and other graphic features. Thinkfree Show opens, edits and saves directly into the Microsoft PowerPoint (.ppt) format.

CyberdrivePlus

A free, one-year subscription to CyberdrivePlus is also included. CyberdrivePlus provides you with secure, Internet file storage and free online software upgrades!

Macworld

"Thinkfree is a best-of-breed program that will exceed your expectations."
— *Jeffery Battersby*

MacAddict RATED GREAT

"Thinkfree Office is the next best thing and then some."
— *Deborah Shadovitz*

macHOME

"Thinkfree Office is an impressive attempt to crack the seemingly impenetrable productivity market."
— *Chris Ward*

amazon.com.
Apple Store
Apple Specialist
COMPUSA.com
Fry's ELECTRONICS
MAC Solutions "Macintosh Superstore" 1-800-873-3726
MacMall
JR
Small Dog Electronics
ZONES

ONLY

$49.95

Figure 3. *AppController inspector shows the four actions we created.*

We've now defined a template for objects of the class AppController. To actually work with such an object here in IB, we need to create one. Choose Classes ⑧ Instantiate AppController to make an AppController object that will be stored in the nib file.

Now it's time to wire this sucker up. IB's famous shortcut for connecting one object to another is to hold down the Control key while dragging between them. We're going to Control-drag from the object that will send the message (in this case, each of the buttons) to the object that will perform the action (the AppController). We hold down Control and start dragging from the first button to the MyDocument.nib window, which switches to the Instances tab as soon as we drag in, and we let go of the mouse button when we've drawn the line to the AppController object. After we connect the two objects, the inspector shows the possible actions in AppController that we can connect the button to (see **Figure 4**). Pick the appropriate action (such as appModalOld for the first button) and double-click to make the connection. We repeat this process for each of the four buttons.

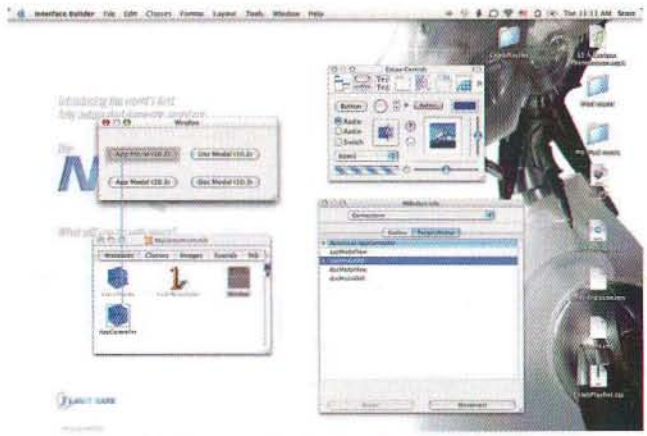


Figure 4. *Connect the buttons to the AppController by Control-dragging in Interface Builder.*

We can use the inspector to tweak the settings of our objects. For example, we can prevent the window from having a resize box in the lower right corner by inspecting it, going to the Attributes page, and turning off the "Resize" setting.

Now we've built a window with four buttons, created the AppController class, made an instance of that class, and connected the buttons to actions in AppController. The nib file contains all the instances we built: the window and its buttons, along with the AppController. Before we leave IB, it will do one final favor for us. We go back to the MyDocument.nib window and click the Classes tab. Making sure AppController is selected, we choose Classes ⑧ Create Files for AppController. This generates skeleton header and implementation (.h and .m) files for our new class and adds them to the project. If only it would write all the code for us, we would be done. But for now, we still have to do that part ourselves. We'll bid adieu to our friend Interface Builder, save the nib file, and go over to Xcode.

Xcode Marks the Spot

In Xcode, we're going to flesh out the four alert-posting methods of AppController that we hooked up to the buttons in IB. The buttons on the left side of the window put up application modal alerts, while the two on the right produce document-modal sheet alerts. Let's talk about the app modal alerts first.

In versions of Mac OS X before 10.3, you put up an app modal alert by calling the function NSRunAlertPanel. Note that this is a plain old C function call – no messages sent to objects. NSRunAlertPanel takes parameters that let you specify its text and buttons, and returns an integer that indicates which button the user clicked. Here's the full syntax:

```
int NSRunAlertPanel(NSString *title,    // main alert text
 NSString *msg,  // second line of alert text
 NSString *defaultButton,    // right button (usually 'OK')
 NSString *alternateButton,  // left button ('Don't ...')
 NSString *otherButton, // middle button (Cancel)
 ...) // printf-style formatting for strings
```

You don't have to pass values for all the parameters – some of them get default values, and others are optional. If you pass nil for defaultButton, the alert will use the localized version of "OK" for the button label. The other buttons are optional – if you pass nil, they don't appear. Passing nil for title gets the localized word for "alert".

Here's the code for putting up the alert using NSRunAlertPanel, which works with all versions of Mac OS X:

```
- (IBAction)appModalOld:(id)sender
{
    NSLog (@"appModalOld");
    int choice = NSRunAlertPanel
(@"Do you want to revert to factory settings?",
 @"If you revert to factory settings, you will lose any cool
settings of your own that you have made. (10.2 version)",
 @"Revert", @"Don't Revert",@"Cancel");
    NSLog (@"NSRunAlertPanel returned %d", choice);
}
```

When you call NSRunAlertPanel, you get some user keyboard shortcuts (also known as key bindings) automatically. If the user presses Return or Enter, that's the same as clicking the first button. Pressings Esc is a shortcut for clicking Cancel. If you have a button labeled "Don't Save", the user can type Command-D as a shortcut.

With the advent of Mac OS X 10.3, Cocoa added a groovy new objectve way to put up alerts: the class NSAlert. To use NSAlert, you create a new alert object, set it up by sending messages to it, then put it on the screen by calling its runModal method. Like NSRunAlertPanel (the pre-10.3 function we used above), runModal returns an integer that tells which button the user clicked. Here are some of the NSAlert methods you're likely to call when setting up an alert:

```
// Set the main text for the alert
 (void) setMessageText:(NSString *)messageText

// Set the secondary text for the alert
 (void) setInformativeText:(NSString *)infoText

// Choose Warning, Info, or Critical alert
 (void) setAlertStyle:(NSAlertStyle) style

// Add a button to the alert
 (NSButton *) addButtonWithTitle:(NSString *)aTitle
```

Calling addButtonWithTitle returns the button, although you'll often ignore the return value when you're creating the alert. You can call methods on the buttons to tweak their appearance and behavior. For example, you can call setKeyEquivalent to make a keyboard shortcut, or setImage to put a custom image on the button.

When you have the alert built to your specifications, you send it the runModal message and the alert appears. When the user dismisses the alert, runModal returns a value that tells you which button the user clicked. Here's how to put up the same alert as we did before, but using the 10.3 technique with NSAlert:

```
- (IBAction)appModalNew:(id)sender
{
    NSLog (@"appModalNew");
      // This requires 10.3. We could check
      // NSAppKitVersionNumber at this point to be sure
      // it's supported.

      // Create the alert object
    NSAlert *theAlert = [[NSAlert alloc] init];

  // Make the buttons.
  // For easiest localization, we could
  // put these strings in a property list and
  // read them in.
  // Buttons are created from right to left.
  //
[theAlert setMessageText:@"Do you want to revert to factory
settings?"];
[theAlert setInformativeText:@"If you revert to factory
settings, you will lose any cool settings of your own that
you have made. (10.3 version)"];
    [theAlert setAlertStyle:NSWarningAlertStyle];
    [theAlert addButtonWithTitle:@"Revert"];
    [theAlert addButtonWithTitle:@"Don't Revert"];
    [theAlert addButtonWithTitle:@"Cancel"];
  // addButtonWithTitle actually returns the button object,
  // so you can mess with buttons using code like this:
  // NSButton *aButton =
  //     [theAlert addButtonWithTitle:@"Revert"];
  //     [aButton setKeyEquivalent:@"r"];
 //    [aButton setBezelStyle:NSThickerSquareBezelStyle];
  //     [aButton setSound:mySound];
  //  etc.

  // Put up the alert and report the choice
int choice = [theAlert runModal];
    switch (choice)
    {
        case NSAlertFirstButtonReturn:  NSLog
(@"NSAlertFirstButtonReturn"); break;
        case NSAlertSecondButtonReturn:  NSLog
(@"NSAlertSecondButtonReturn"); break;
        case NSAlertThirdButtonReturn:  NSLog
(@"NSAlertThirdButtonReturn"); break;
    }
    NSLog (@"NSAlert runModal: returned %d", choice);


  //This works too:"compatibility" method.
  // Note returned button values are different (-1, 0, 1 instead of 1000, 1001, 1002).

    /*
   theAlert = [NSAlert alertWithMessageText:@"Do you want to
revert to factory setting?"
                defaultButton:@"Revert"
                alternateButton:@"Don't Revert"
                  otherButton:@"Cancel"
          informativeTextWithFormat:@"If you revert to
factory settings, you will lose any cool settings of your own
that you have made. (10.3 version)"];
   choice = [theAlert runModal];
   NSLog (@"NSAlert runModal: returned %d", choice);
   */

}
```

Mac OS X 10.3 also provides a "compatibility" method that looks a lot like the NSRunAlertPanel function. There's an example of it commented out at the bottom of the last listing. This is implemented as a class method of NSAlert. Here's its declaration:

```
+ (NSAlert *)alertWithMessageText:(NSString *)messageTitle
    defaultButton:(NSString*)defaultButtonTitle
    alternateButton:(NSString *)alternateButtonTitle
    otherButton:(NSString*)otherButtonTitle
    informativeTextWithFormat:(NSString *)format, ...
```

There's other customizing you can do with NSAlert, including adding help (setShowsHelp) by using a delegate object. Check the documentation for NSAlert (and NSButton) to learn about more cool tweaks.

Changing the Sheets

Now we're going to write methods that put up document modal alerts, also called sheets. These are the cool eye-candy dialogs that slip out from window title bars, then roll back in when you're done. As with our previous examples, there are old-school and new-wave ways to do this. We'll show both.

For best compatibility with previous and current versions of OS X, use the function NSBeginAlertSheet, defined as follows:

```
void NSBeginAlertSheet(NSString *title,
  NSString *defaultButton, // define three buttons
  NSString *alternateButton,
  NSString *otherButton,
  NSWindow *docWindow, // window the sheet goes with
  id modalDelegate, // sheet's delegate object
  SEL didEndSelector, // called after user finishes
  SEL didDismissSelector, // called when sheet is gone
  void *contextInfo, // passed to delegate as data
  NSString *msg, ...)
```

The first few parameters are similar to those we used for app modal alerts earlier, but there's interesting new stuff too. We get to supply pointers to a couple of methods, one that gets called when the user clicks a button to end the alert, and the other that's summoned after the alert is actually closed. We can use these to validate data, record what happened, or for any other purpose. The following listing shows a barebones example of how to put up the document modal alert in our little sample:

```
- (IBAction)docModalOld:(id)sender
{
    NSLog (@"docModalOld");

  NSBeginAlertSheet (@"Do you want to revert to factory
settings? (10.2 version)", // title
            @"Revert", @"Don't Revert",@"Cancel",
          [sender window], // sender is a button. We can
                // get its window this way
              nil, // delegate (owner of next two methods)
              nil, // if this is a method selector,
                // it's called when sheet is done
              nil, // if this is a method selector,
                // it's called when sheet is closed
              nil, // this pointer is passed to the
                // delegate when previous two
                // methods are called
              @""); // message to display in sheet
                // (Note: if this is nil, we get
                // a runtime error.)
}
```

In this example, we're passing nil and so skipping the chance to use the didEndSelector and didDismissSelector hooks. If you want to use either of these methods, define them like this:

```
sheetDidEnd:(NSWindow *)sheet
    returnCode:(int)returnCode
    contextInfo:(void *)contextInfo;

sheetDidDismiss:(NSWindow *)sheet
    returnCode:(int)returnCode
    contextInfo:(void *)contextInfo;
```

When you call NSBeginAlertSheet, pass a pointer to your methods as the appropriate parameter, like this:

```
@selector(sheetDidEnd:returnCode:contextInfo:)    and
@selector(sheetDidDismiss:returnCode:contextInfo:)
```

Cocoa will call your sheetDidEnd method when the user finishes with the sheet, and the sheetDidDismiss method after the sheet is actually gone. You can arrange to pass anything you want in the contextInfo parameter.

Just as with app modal alerts, the freshest Mac OS X 10.3 Cocoa provides the NSAlert class for you to create document modal alerts. You use it by sending messages to build an NSAlert object, then sending another message to actually show the alert. Here's the relevant code from our sample app:

```
- (IBAction)docModalNew:(id)sender
{
    NSLog (@"docModalNew");

  NSAlert *theAlert = [[[NSAlert alloc] init] autorelease];
    // set up buttons
  [theAlert addButtonWithTitle:@"Revert"];
  [theAlert addButtonWithTitle:@"Cancel"];
  [theAlert addButtonWithTitle:@"Don't Revert"];

    // set up text
  [theAlert setMessageText:@"Do you want to revert to factory
settings?"];
  [theAlert setInformativeText:@"If you revert to factory
settings, you will lose any cool settings of your own that
you have made. (10.3 version)"];

    // choose alert kind
  [theAlert setAlertStyle:NSWarningAlertStyle];

    // roll out the sheet!
    // We're not using a delegate or its hook. If we were,
    // we would define a method called
    // alertDidEnd:returnCode:contextInfo  and
    // pass @selector(alertDidEnd:returnCode:contextInfo:)
    //  for didEndSelector and self for modalDelegate
  [theAlert beginSheetModalForWindow:[sender window]
        modalDelegate:nil
        didEndSelector:nil
        contextInfo: nil];

}
```

This completes the basic framework of our sample app. Note that because we've created a document-based application, we can open as many windows as we want, and each one comes with sheet-displaying abilities. For fun, open a whole mess of them and drag them around. Whee!

As with all software, there are a jillion ways we could enhance our little app (such as by making it do some actual work instead of just demonstrating how to put up alerts). For example, we could take the English text out of the code and put it in a property list, which would be better for localization. We might try a customized sheet, which we can make with the beginSheet:modalForWindow:modalDelegate:didEndSelector:contextInfo: method of NSApplication. Delve into the definitions and documentation for NSAlert, NSApplication, and NSButton for more exploration. Extending this example is a great way to get started playing around with Cocoa. You can download the Xcode project from http://www.papercar.com/mt/RedAlert.zip.

Until next month, have fun, stay alert, and please yell (scottk@mactech.com) if you are paying attention.

Andrew Troelsen, Minneapolis, Minnesota

A Primer on the C# Programming Language (Part One)

Exploring .NET development on Mac OS X

Topic Prelude

If you have read the previous installments of this series (except last month when I missed the deadline), you should be well aware that the .NET platform is language agnostic. While this is technically correct, I'd be lying if I did not admit that C# is currently the language of choice for a majority of .NET developers. Given this fact, my goal in the next two issues is to offer a tour of the major syntactical features provided by C#. Understand of course, that two articles cannot possibly examine each and every token and construct of a given language. In any case, once you have digested the material presented here, you should feel quite comfy with the core aspects of C# and be in a perfect position to follow along with the articles to come.

Recall the Pillars of OOP

All object languages must contend with the three 'pillars of object oriented programming' (OOP). First, an OOL (object oriented language) must address the concept of *encapsulation*. Simply put, encapsulation services provide a way for types to safely control access to their private data points. Next we have *inheritance*, which provides a manner in which exiting types may be reused and extended. Last but not least we have the third pillar of OOP termed *polymorphism*. This pillar of OOP allows related types to respond uniquely to the same method invocation. Given that C# is a modern OOL, you are correct to assume that C# provides complete support for the pillars of OOP, as well as a relatively new paradigm termed *aspect oriented programming* (AOP) which will be examined in a later article. Before we dig into the details of OOP using C#, let's check out basics of defining classes and allocating objects.

Defining Class Types in C#

Classes are the cornerstones of any OOL, in that they provide a template for the objects that comprise your application. In C#, class types are defined using (not surprisingly) the 'class' keyword. Like other languages in the C family, C# classes may support any number of overloaded constructors to allow an object to come to life in a particular state. To illustrate, assume we have a class named Car (defined in a file named car.cs), which defines three pieces of private data to represent a given automobile's pet name, current speed and make (**Listing 1**).

Listing 1. A simple C# class definition (car.cs).

```
using System;

namespace CSharpOOPExample
{
  //The Car class type.
  public class Car
  {
    // State data (could be defined as protected if desired).
    private string petName;
    private int currSpeed;
    private string make;

    // Constructor set.
    public Car(){}
    public Car(string pn, int cs, string m)
    {
      petName = pn; currSpeed = cs;
      make = m;
    }

    // Override the virtual method System.Object.ToString()
    // in order to display our state data.
    public override string ToString()
    {
      return string.Format(
        "Name: {0}; Speed: {1}; Make: {2}", petName,
        currSpeed, make);
    }
  }
}
```

Andrew Troelsen is a seasoned .NET developer who has authored numerous books on the topic, including the award winning *C# and the .NET Platform*. He is employed as a full-time .NET trainer and consultant for Intertech Training (www.intertechtraining.com), but thinks he is still on the beaches of Mexico sipping funky drinks served in coconut shells. You can contact Andrew at atroelsen@mac.com.

Multiple formats. Multiple platforms. Complex installers.

Aladdin solves the compression and installation puzzle.

Trying to figure out how to handle multiple compression formats and platforms?
The StuffIt Engine solves the compression puzzle.

Aladdin's StuffIt Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the StuffIt file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low as $99/year

To learn more, visit: www.stuffit.com/sdk/

StuffIt Engine SDK™
The power of StuffIt in your software.

Looking for the easiest and fastest way to build an installer?
StuffIt InstallerMaker completes your puzzle.

It's not enough just to write solid code anymore. You still have to write an installer for your users. StuffIt InstallerMaker makes it simple and effective.

- StuffIt InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with StuffIt InstallerMaker.

Prices start at $250

To learn more, visit: www.stuffit.com/installermaker/

StuffIt InstallerMaker™
The complete installation solution.™

Aladdin Systems

www.stuffit.com
(831) 761-6200

The first point of interest has to do with the 'public' keyword used when defining the type. In C#, a non-nested type may be defined as public or 'internal' (the default). The distinction is made clear when you recall that types are contained within an assembly. Public types may be created and manipulated across assembly boundaries. On the other hand, internal types can only be created and used by the assembly that defines it. Thus, if you were to build a .NET assembly which contained three public types and two internal types, other assemblies would be completely unaware of the two internal types. Using this technique, assembly builders are free to define any number of internal helper types that are invisible to other binary units.

Next, you can see that we have defined a method named ToString() using the C# 'override' keyword. This brings up a very important point regarding the .NET platform: If you do not say otherwise, class types automatically derived form the cosmic base class, System.Object (introduced in the October 2003 installment of *Casting your .NET*). Like many base classes, Object defines a set of virtual methods that may be redefined (which is to say, *overridden*) by derived classes. Here, the virtual System.Object.ToString() method has been overridden in order to return a custom blurb of text which represents the current state of a given Car type (the default implementation of ToString() returns the type's fully qualified name). We'll examine the details of virtual members and member overriding a bit later in this article.

Finally, note that the Car class has two constructors. The role of our custom constructor is clear enough, in that it allows the object user to create a car in an initial state. The other constructor may seem a bit odd (depending on your exposure to other C based languages) in that it's implementation does nothing at all. C#, like Java and C++ always supply a freebee no-argument constructor (termed the default constructor) for each and every class definition. However, as soon as you define a custom constructor, the default, default constructor (pardon the redundancy) is removed. Therefore, if you with to allow objects of this type to be created with no arguments, you must redefine the no-argument constructor.

On a final constructor-related note, recall the C# automatically sets a type's state data to safe default values (quite *unlike* C++). The rules are quite simple:

- Numerical data is set to 0 or 0.0.
- Object references are set to null.
- Booleans are set to false.

Given this language feature, we have no need to assign values to petName, currSpeed or make within the scope of the default constructor.

Allocating and (indirectly) Destroying Objects

We are now ready to create an application object (e.g., the class defining the Main() method) to allocate an auto or two onto the managed heap (**Listing 2**).

Listing 2. Creating Cars (carApp.cs).

```
using System;

namespace CSharpOOPExample
{
  // The App object.
  internal class CarApplication
  {
    // Program's entry point
    private static void Main()
    {
      // Make some Cars!
      Car noName = new Car();
      Console.WriteLine(noName.ToString());

      Car someCar = new Car("Zippy", 90, "BMW");

      // ToString() called automatically.
      Console.WriteLine(someCar);
    }
  }
}
```

In C#, the 'new' keyword is used to allocate an instance of a given class type onto a region of memory termed the *managed heap* (in fact, class types can only be heap allocated. Stack allocated data is achieved through the use of value types, as described in the next issue). In a similar fashion as the Java platform, .NET provides a garbage collector that is in charge of destroying unused objects when required. Given this, C# does not provide a corresponding keyword (such as 'delete') to explicitly destroy an object. However, C# does provide a syntax which looks suspiciously close to a C++ style destructor. Ponder the following updated Car type (**Listing 3**):

Listing 3. C# style 'destructors'.

```
// The updated Car class type
public class Car
{
  ...
  ~Car()
  {
    Console.WriteLine("This car is destroyed.");
  }
}
```

Like C++, C# destructors are syntactically denoted using a tilde prefix, however the similarities end here. First and foremost, a C# destructor is in reality a short hand notation for overriding the virtual System.Object.Finalize() method (to verify this for yourself, run your assembly through ildasm and check out how C# destructors are expressed in terms of CIL code). Next, remember that C++ memory management is quite deterministic in nature, in that we programmers are the individuals responsible for destroying heap allocated memory. In contrast, the .NET platform uses a non-

deterministic finalization approach. In other words, you do not know exactly *when* the garbage collector will remove an object from the heap, only that the associated memory will eventually be reclaimed. When the object is indeed destroyed, the .NET runtime will ensure that your type's destructor is invoked if present.

As you might guess, there is much more that could be said about the .NET garbage collection process (such as programmatically controlling garbage collection using System.GC, implementing the IDisposable interface, object generations, etc), however the current explanation will suffice for now.

Encapsulation Support via Type Properties

Our current iteration of the Car type is dysfunctional to say the least, as we have not provided a way to get or set the state data after the time of construction! .NET programming languages prefer the use of type properties, rather than traditional getters and setters to honor the pillar of encapsulation. In the February 2004 issue you examined property syntax via JScript.NET, and will be happy to know that the act of defining properties in C# is quite similar. Listing 4 illustrates how to preserve encapsulation of the private petName member variable using a public property named Name (assume that the make and currSpeed member variables are encapsulated by additional properties; Make and Speed respectively).

Listing 4. C# property syntax.

```
public class Car
{
  private string petName;
...
  // C# Property syntax.
  public string Name
  {
    get { return petName; }
    set { petName = value; }
  }
}
```

Recall that the name of a property does not need to have any relationship to the name of the data point it is responsible for exposing. Also note the use of the 'value' token in the implementation of the property set scope. Truth be told, 'value' is not really a true-blue keyword in the C# programming language given that it is legal to define member variables or local variables named 'value'. However when this token appears in the context of a property setter, it is used to represent the incoming value assignment. Finally, it is worth noting that properties can be configured as read-only or write-only. Simply omit the get or set scope from the property definition.

For testing purposes, update your Main() method to change and obtain various data points (**Listing 5**).

Listing 5. Exercising our properties.

```
internal class CarApplication
{
  // Program's entry point
  private static void Main()
  {
    ...
    Car someCar = new Car("Zippy", 90, "BMW");
    someCar.Name = "Junior";
    Console.WriteLine("{0} is going {1} MPH.",
      someCar.Name, someCar.Speed);
  }
}
```

At this point you can compile your C# files into an executable assembly (don't forget to enable an SSCLI-aware Terminal). The commands listed in **Listing 6** will do the job nicely.

Listing 6. Compiling and executing our Car application.

```
DoSscli
csc *.cs
clix carApp.exe
```

Next up, we will examine how to build a set of related types using classic inheritance.

The Syntax of Inheritance

As mentioned, if you build a class type that does not explicitly specify a base class, your type will automatically derive from System.Object. However, when you wish to build class hierarchies you will no doubt be interested in deriving new types from existing class definitions. This is accomplished in C# using the colon operator. Let's create three child classes (SportsCar, MiniVan and JamesBondCar) that leverage our current Car type (assume each of these new types are within a file named childCars.cs and are wrapped within the CSharpOOPExample namespace). **Listing 7** defines the SportsCar type.

Listing 7. The SportsCar class type (childCars.cs).

```
// SportsCar IS-A Car.
public class SportsCar : Car
{
  public SportsCar(string pn, int cs, string m)
    : base(pn, cs, m) {}
  public SportsCar(){}

  public void TurboBoost()
  { Speed =+ 20; }
}
```

Beyond the fact that SportsCar is explicitly deriving from the Car type (and therefore inherits each of the public and protected members of it's base class), observe the use of the 'base' keyword in the custom constructor. As you can see, 'base' is dangling off the constructor definition by way of a single colon (which in this case is not marking the name of the base class). When the base keyword is used in this manner, you are specifying which constructor to call on the parent class when the derived type is created. Given that Car already has storage for the petName, currSpeed and make data points, we are explicitly calling the three-argument constructor of the parent. If we did not do so, the parent's default constructor would be called automatically,

forcing us to set the private data points using the inherited public properties or possibly protected data (which is obviously less efficient).

As well, notice the implementation of the TurboBoost() method. Another bonus of property syntax is that they respond to the intrinsic operators of C#. Thus, rather than having to increase the SportsCar's speed using traditional accessor and mutator logic (**Listing 8**) we can use the more streamlined code "Speed += 20;".

Listing 8. Properties streamline traditional get / set logic.

```
// If we were not using properties...
public void TurboBoost()
{
  setSpeed(getSpeed() + 20);
}
```

Listing 9 details the MiniVan type, who's first point of interest is that the custom constructor passes three of the four incoming arguments to the base class, while assigning the fourth and final parameter to it's own custom point of data (numberOfKids). Also notice how MiniVan overrides the parent's implementation of ToString() to account for it's custom piece of state data. In this case, the 'base' keyword is *not* triggering a base class constructor using the 'dangling colon on the constructor' syntax, but simply calling a base class method within the scope of the method definition.

Listing 9. The MiniVan class type (childCars.cs).

```
// MiniVan IS-A Car.
public class MiniVan : Car
{
  public MiniVan(string pn, int cs,
    string m, int k)
    : base(pn, cs, m)
  {
    numberOfKids = k;
  }
  public MiniVan(){}

  private int numberOfKids;
  public int KidCount
  {
    get {return numberOfKids;}
    set { numberOfKids = value;}
  }

  public override string ToString()
  {
    return string.Format("{0}; kids: {1}",
      base.ToString(), numberOfKids);
  }
}
```

Finally, **Listing 10** illustrates the final automobile, JamesBondCar. Note that JamesBondCar extends SportsCar, which in turn extends Car (which extends System.Object).

Listing 10. The JamesBondCar class type (childCars.cs).

```
// JamesBondCar IS-A SportsCar
// which IS-A Car.
public class JamesBondCar : SportsCar
{
  public JamesBondCar(string pn, int cs, string m)
    : base(pn, cs, m) {}
  public JamesBondCar(){}

  public void DiveUnderWater()
  { Console.WriteLine("Diving under water!");}
  public void Fly()
  { Console.WriteLine("Taking off into the air!");}
}
```

Needless to say, our JamesBondCar is able to dive under water and fly into the air to escape the current enemy at hand. Further, given that this type does not add any new member variables to the mix, the parent's ToString() implementation will fit the bill nicely.

Building a Array of Car types

At this point the Main() method may be updated to exercise each of these new derived classes. To provide a more interesting example however, let's create an array of Car types. As you are most likely aware, most OOLs (including Java and C++) allow you to store a derived object in a base class reference (e.g., and implicit cast). This is legal given the 'IS-A' relationship enforced by classical inheritance. Given this fact, update Main() to create an array of Car-compatible types and iterate over the array using the C# 'foreach' keyword to invoke each object's ToString() implementation (**Listing 11**).

Listing 11. Creating an array of Car-compatible types.

```
// Make an array of Car types.
Car[] allTheCars = new Car[3]
 { new SportsCar("Zippy", 85, "Audi TT"),
   new MiniVan("KidMobile", 55, "Caravan", 10),
   new JamesBondCar("QMobile", 120, "*Classified*")};

// Print out the number of cars in array.
Console.WriteLine("You have {0} cars:",
  allTheCars.Length);

// Call each auto's ToString() method.
foreach(Car c in allTheCars)
 Console.WriteLine(c.ToString());
```

A few points of interest. In C#, arrays are declared using the C-like square bracket notation. Here we have created an array of Car types named allTheCars. The 'new' keyword used when declaring the array is *not* creating any particular Car type, but rather the underlying System.Array in the background. Given that C# arrays always derive from the System.Array base class, each array has access to each of the public members (such as the Length property seen in the previous code segment).

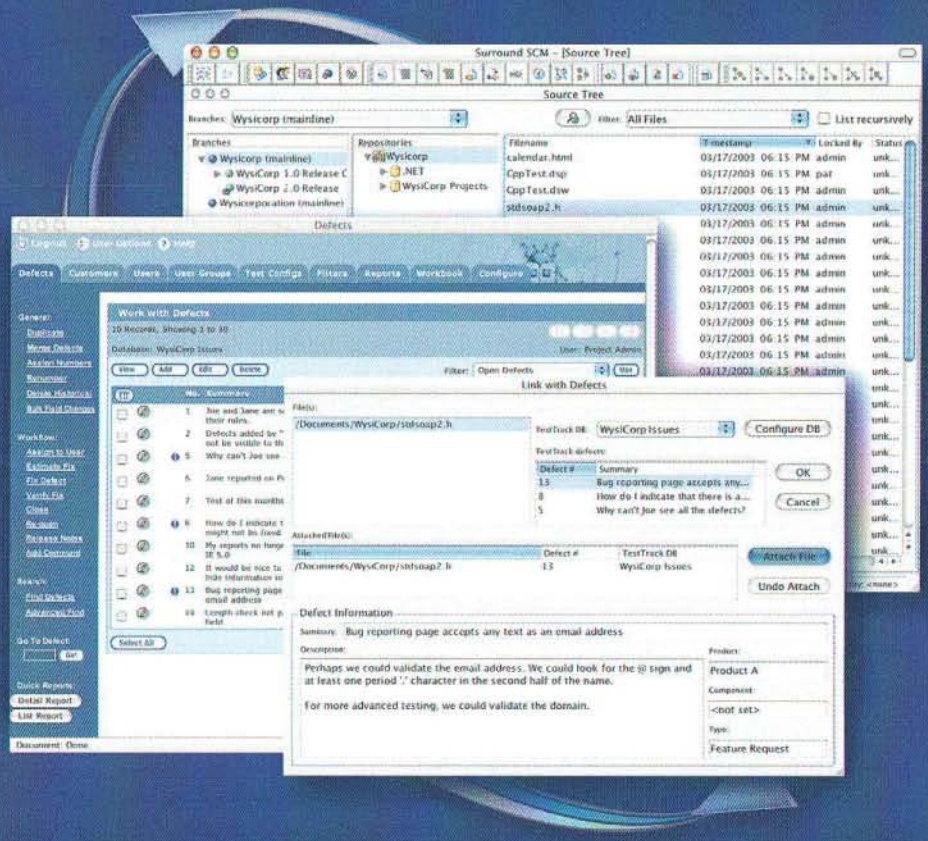
Once we have allocated a System.Array capable of holding Car-compatible types, we can leverage the curly-bracket shorthand notation to fill the array with sub-elements at the time of creation. If you would rather, you are free to allocate and initialize an array on an item-by-item basis (**Listing 12**).

Listing 12. Creating an array of Car-compatible types (the long way).

```
// Long hand array creation.
```

Complete Source Control and Defect Management for Mac OS X

Seapine Software™
changing the world of software development



Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

Seapine Software Product Lifecycle Management
Award winning, easy-to-use software development tools

Seapine **Surround SCM**

Seapine **TestTrack PRO**

software development 13th annual product excellence award

Download Surround SCM and TestTrack Pro at www.seapine.com or call 1-888-683-6456

```
Car[] allTheCars = new Car[3];
allTheCars[0] = new SportsCar("Zippy", 85, "Audi TT");
allTheCars[1] =
  new MiniVan("KidMobile", 55, "Caravan", 10);
allTheCars[2] =
  new JamesBondCar("QMobile", 120, "*Classified*");
```

The Details of C#'s 'foreach' Keyword

Speaking of System.Array, if you were to look up the formal definition of System.Array using any of the tools that ship with the SSCLI (see Feb 2004) you will find that this class type implements an interface named System.Collections.IEnumerable. System.Collections.IEnumerable defines a single method named GetEnumerator(), which returns yet another interface named System.Collections.IEnumerator. This interface provides a way for a type to iterate over contained sub-items using three members:

- Current : This property returns the item current 'pointed to'.
- Reset() : This method resets the internal indexer's position to the first item.
- MoveNext() : As you would guess, this advances in the internal indexer by one. Returns false when the end of the list has been reached.

So, why do we care about IEnumerable and IEnumerator? Well, the 'foreach' keyword of C# is preprogrammed to obtain these underlying interfaces to traverse the sub-objects of the array being iterated over. Given that all of these sub-objects have a ToString() implementation, we can safely invoke each auto's custom version. Be aware that System.Array is not the only type which implements the necessary interfaces required by the foreach construct. Most of the class types found within System.Collection support similar infrastructure, and if you wish to build a strongly typed custom collection, you can implement these interfaces directly to traverse custom types using 'foreach'.

The ABCs of Polymorphism: Virtual and Abstract Members

The final pillar of OOP to examine is polymorphism, which you have already begun to leverage when you overrode the virtual System.Object.ToString() and System.Object.Finalize() methods in your custom class types. As mentioned, classical polymorphism is the trait of OOP that allows hierarchies of types to responding uniquely to the same message (a.k.a., method invocation). Polymorphism is supported in C# using four simple keywords:

Base: As already seen, 'base' allows you to trigger a parent's method / constructor.

Virtual: This keyword allows you to define a base class method that has a default implementation, but may be overridden by a child class if required.

Abstract: This keyword allows you to define an abstract base class (ABC) as well as abstract methods. Recall that abstract classes cannot be directly created, but can be used to hold references to derived types. Also recall that abstract methods do *not* have a default implementation, and therefore derived types must provide a concrete implementation, or define themselves as abstract classes as well.

Override: This keyword allows a derived class to redefine a virtual method as well as implement an abstract member.

To inject some polymorphic activity into our existing application, let's add an abstract method to our Car base class called PrintBumperSticker(). Be aware that when a class defines an abstract member, the class must also be marked as abstract (**Listing 13**).

Listing 13. The abstract Car type.

```
//The abstract Car class type
public abstract class Car
{
  // All derived classes must
  // implement this member or become
  // abstract types as well.
  public abstract void PrintBumperSticker();
...
}
```

Given that Car has now been redefined as an ABC, it is a compile time error to directly create Car types. However Car (and ABCs in general) still has a very useful purpose, in that it defines all of the common functionality for derived types. Car has been further updated with a single abstract member, and therefore each and every derived class in now required to provide an implementation of this member (if you do not, you are issued compile time errors).

To rectify this issue, update SportsCar, MiniVan and JamesBondCar as you see fit using the 'override' keyword. **Listing 14** shows one possible implementation for the MiniVan type.

Listing 14. MiniVan's PrintBumperSticker() implementation.

```
public class MiniVan : Car
{
  public override void PrintBumperSticker()
  {
    Console.WriteLine
     ("All my money and Kids go to the U of Mn.");
  }
...
}
```

Runtime Type Discovery using C#

Now that we have a polymorphic interface defined by our base class, we can update our Main() method to invoke each type's custom implementation (**Listing 15**).

Listing 15. Polymorphism at work.

```
foreach(Car c in allTheCars)
{
  Console.WriteLine(c.ToString());
```

```
  c.PrintBumperSticker();
  Console.WriteLine();
}
```

Understand that the code within the foreach block is only legal because of the defined polymorphic interface. We can rest assured that all types have an implementation of ToString() given the fact that all classes ultimately derive from System.Object, and that all descendents of car must implement the abstract PrintBumperSticker() method. However, what if we wish to call specific members of the JamesBondCar, MiniVan or SportsCar types? If you look closely at the foreach syntax, you can see that we are using a base class reference to represent each sub-object. Therefore, the following would be a compile time error (**Listing 16**).

Listing 16. Can't directly access derived type members from a base class reference!

```
foreach(Car c in allTheCars)
{
  ...
 // Nope! Car does not define a KidCount
 // property! Compiler error!
  int numberOfKids = c.KidCount;
}
```

When you need to dynamically discover if a given type is comparable with a given base class (or interface), C# provides two keywords. Ponder the following update (**Listing 17**).

Listing 17. Runtime type discovery in C#

```
foreach(Car c in allTheCars)
{
  Console.WriteLine(c.ToString());
  c.PrintBumperSticker();

 // Is 'c' a JamesBondCar?
  if(c is JamesBondCar)
    ((JamesBondCar)c).DiveUnderWater();

 // Is 'c' a MiniVan?
  MiniVan m = c as MiniVan;
  if(m != null)
    Console.WriteLine("I have {0} screaming kids!",
      m.KidCount);
  Console.WriteLine();
}
```

The 'is' keyword is quite helpful in that it returns a System.Boolean that denotes if the current object is compatible with a given base class (or interface type). If the test succeeds, you can make a safe explicit cast (using the familiar C-style casting syntax) to access the underlying functionality. The 'as' keyword is similar, however 'as' will return a null object reference if the types being tested are incompatible. Therefore, when performing a runtime check for type compatible using the 'as' keyword, be sure to test for null before casting!

On a final casting related note, you can make use of one additional construct to check for type compatibility: structured exception handing (**Listing 18**).

Listing 18. Manually handling an InvalidCastException.

```
// Is 'c' SportsCar compatible?
try{
  ((SportsCar)c).TurboBoost();
}
catch(InvalidCastException ex)
{
  Console.WriteLine("OOPS!  Not a SportsCar.");
  Console.WriteLine(ex.Message);
}
```

Here, we are making use of structured exception handling to attempt to cast current item pulled from the array of Car-compatible types into a SportsCar. If the cast fails, the runtime will throw a System.InvalidCastException type. Given that all exceptions derive from a common base class named System.Exception, we are free to make use of any of the inherited members to display information about the error in question (such as the Message property). See online help for complete details of System.Exception.

Interface-Based Polymorphism

Excellent! At this point you have a solid understanding of how C# contends with the mighty pillars of OOP and gained some insights into runtime type discovery, explicit casting and structured exception handling along the way. To complete this article, we will shift away from our examination of class types and examine the role of interface types (remember that .NET defines five possible types from the set {class, interface, structure, enumeration, delegate}).

Interface types, in a nutshell, are a named collection of abstract (and only abstract) members. On its own, an interface is of little use, given that you cannot create an instance of an interface variable. However, when an interface is implemented on a given class (or structure) you are able to bind a set of behaviors to the type in question. The power of interface-based programming becomes crystal clear when you understand that these types allow you to breath polymorphism into types found in *different class hierarchies*. Let's see a complete example.

Assume you have created some new class type modeling UFOs. Given that UFOs (presumably) are not automobiles, the UFO base type will derive directly from System.Object, whereas MotherShip extends UFO (**Listing 19**).

Listing 19.The UFO class types (ufos.cs).

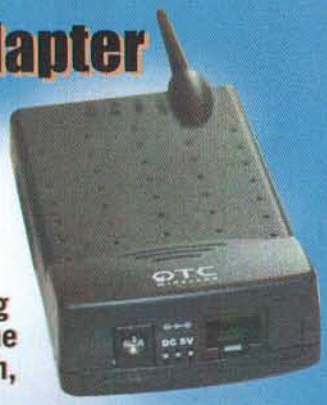
```
namespace CSharpOOPExample
{
  public class UFO
  {
    public void AbductHuman()
    { Console.WriteLine("Come here Earthling...");
  }

  public class MotherShip : UFO
  {
    public void AbductOtherUFOs()
    { Console.WriteLine
       ("You have violated the prime directive."); }
```

Extreme to the Macs!

Aria Extreme
$78.95

Add AirPort Extreme to any G3 or G4 PowerBook! Get blazing speeds up to 54 Mbps (five times AirPort Speed!), comes with an external stub antenna for better range than internal cards! (requires Mac OS X 10.2.6 or above)

OTC Wireless 802.11g Ethernet Adapter
$164.95



Driverless Ethernet to 802.11g converter. Simply plug into the Ethernet port of any Macintosh, PC, or even printer to put it transparently on your 802.11g (54 Mbps) network! Use with a hub to enable multiple devices with one adapter!

DEV DEPOT®

Use AirPort Extreme with any Macintosh!

www.devdepot.com

877-DEPOT-NOW

Toll Free: 877-337-6866 • Outside US/Canada: 805-494-9797 • Fax: 805-494-9798 • orders@devdepot.com
PO Box 5200 • Westlake Village, CA 91359-5200

AirPort Extreme (802.11g) PCI Card

$98.95

Easy to install! Step up to AirPort Extreme (54 Mbps) for any PCI Macintosh! (requires Mac OS X 10.2.6 or above)

AirPort Extreme Omni Antenna 5 dBi

$78.95

Want more range from your AirPort Extreme base station? This 5 decibel antenna is 30% more powerful than other brands!

We also offer a full line of AirPort (802.11b) products!

Mac Wireless USB to 802.11b Converter

$78.95

AirPort PCI to 802.11b Converter

$128.95

AirPort 802.11b PCMCIA Card

$88.95

Belkin Wireless Cable/DSL Gateway Router

$112.95

```
  }
}
```

At this point, we have two distinct hierarchies (car-types and UFO-types). When you look at the classes defined in each category, it might strike you that UFO, MotherShip and JamesBondCar could share common behavior in that they all have the ability to become airborne. If you wish to build a further association between these types, you won't get very far with classical polymorphism given that virtual and abstract methods in an ABC are only useful if the types are in the same hierarchy (which is not the case here). However, if you pull the common functionality into an interface definition, you can selectively attach this behavior on a type-by-type basis. To illustrate, define an interface named IAirVehicle within a file named interfaces.cs (**Listing 20**).

Listing 20. The IAirVehicle interface (interfaces.cs)

```
namespace CSharpOOPExample
{
  public interface IAirVehicle
  {
    void Hover();
    bool CanLeaveAtmosphere {get;}
  }
}
```

Here, IAirVehicle defines a single method and a read only property. Again given that interfaces are nothing but a named collection of abstract methods, we have no implementation details, no access modifier (interface methods are always public) and no member variables.

Once an interface is defined, it may now be implemented by each type that should support this behavior. First, the UFOs (**Listing 21**).

Listing 21. Implementing IAirVehicle on the UFO types.

```
namespace CSharpOOPExample
{
  public class UFO : IAirVehicle
  {
...
  // Mark IAirVehicles.Hover() as virtual
  // to allow derived types to modify this
  // behavior.
    public virtual void Hover()
    { Console.WriteLine
      ("Hovering and observing humans..."); }
    public bool CanLeaveAtmosphere {get {return true;} }
  }

  // Because MotherShip derives from UFO,
  // it is automatically IAirVehicle
  // compatible.
  public class MotherShip : UFO
  {
...
    public override void Hover()
    { Console.WriteLine
      ("Hovering and observing other UFOs..."); }
  }
}
```

Implementing an interface is an all or nothing proposition. Given that UFO states that it supports IAirVehicle, it is now obligated to implement Hover() and CanLeaveAtmosphere. Notice that when UFO does implement the Hover() method, it marks this member as *virtual*. Thus, the derived MotherShip is free to redefine how it will implement this functionality while still being type compatible with IAirVehicle. Now, let's implement this same interface on JamesBondCar (**Listing 22**).

Listing 22. Implementing IAirVehicle on JamesBondCar.

```
public class JamesBondCar : SportsCar, IAirVehicles
{
...
  public void Hover()
  { Console.WriteLine
    ("Observing GoldFinger and OddJob..."); }
  public bool CanLeaveAtmosphere {get {return false;} }
}
```

Note that when you wish to explicitly mark a type's base class as well as implement some set of interfaces, you simply make use of a comma-delimited list (the first item after the semi-colon used on the class definition will always mark the base class). Given that UFO derives directly from System.Object, we can simply list the set of supported interfaces without explicitly deriving from Object (as this is assumed).

Interfaces in Action

At this point, we have three different classes (in distinct hierarchies), which implement the same interface. Now then, what is the benefit of doing so? First of all, you can declare an array of types that implement a given interface to exercise interface-based polymorphism (**Listing 23**).

Listing 23. Interface types as arrays.

```
// Create an array of IAirVehicle compatible types.
IAirVehicle[] myFlyingObjects = new IAirVehicle[4];
myFlyingObjects[0] =
  new JamesBondCar("Bimmer", 120, "*Classified*");
myFlyingObjects[1] = new UFO();
myFlyingObjects[2] = new UFO();
myFlyingObjects[3] = new MotherShip();

foreach(IAirVehicle aVCompatibleType in myFlyingObjects)
{
  Console.WriteLine("Can leave atmosphere? {0}",
    aVCompatibleType.CanLeaveAtmosphere);
}
```

To deepen your appreciation of interface programming techniques, assume we now which to create a System.Collections.ArrayList type within our Main() method. Because the Add() method of the ArrayList type is prototyped to take System.Objects, you can add literally anything into the container (**Listing 24**).

Listing 24. Populating our ArrayList with numerous things...

```
ArrayList allMyStuff = new ArrayList();
allMyStuff.Add(new JamesBondCar());
allMyStuff.Add(22);
```

```
allMyStuff.Add("Go Baldy, it's your birthday...");
allMyStuff.Add(new MiniVan());
allMyStuff.Add(new MotherShip());
allMyStuff.Add(false);

foreach(object o in allMyStuff)
{
  if(o is IAirVehicle)
    ((IAirVehicle)o).Hover();
  else
    Console.WriteLine("sorry, not an air vehicle.");
}
```

Here, the allMyStuff object contains numerous unrelated items (JamesBondCars, System.Int32s, System.String types, System.Booleans and so on), however we are able to investigate each sub-item using the 'is' (or 'as') keyword to dynamically discover which items are IAirVehicle compatable. If the current item is IAirVehicle aware, we cast the object accordingly and call the Hover() method. Again, the beauty of the interface is the fact that we can inject polymorphism across diverse class hierarchies. Furthermore, given the language agnostic nature of .NET, it is commonplace to define an interface in one programming language and implement it within another.

Like garbage collection, there are numerous other topics regarding interface based programming techniques under the .NET platform (explicit interface implementation, interface hierarchies, multiple inheritance of interface types and whatnot), however we'll need to call it a day for the time being.

Wrap Up

In this installment of *Casting Your .NET* you examined how the C# programming language contends with the pillars of OOP. Once you drilled through the basics of class definition and object allocation, you saw how to define type properties in the syntax of C#. Next you created a hierarchy of types using classical inheritance, and injected some polymorphic behavior using abstract methods and interface implementation. In the next article, I'll complete this C# primer by examining the remaining .NET types (enumerations, structures and delegates). See ya next issue, and happy hacking.

By Aaron Montgomery

Documenting your code

Generating code documentation with doxygen on Mac OS X

Introduction

One of the nice features of Mac OS X is that it opens up access to a large number of developer tools that used to be restricted to UNIX and Windows. One such tool is doxygen, an application that aids in code documentation in a manner similar to javadoc. This article describes how to set up doxygen on Mac OS X and assumes that you have only limited UNIX experience. As this article developed, it became more of an introduction to UNIX basics inside the Terminal and less an introduction to doxygen than I originally planned. This is due both to the fact that doxygen already contains a manual and numerous examples (that I found useful), as well as the intended audience (Mac programmers new to UNIX). The article describes how to install doxygen on Mac OS X; how to run doxygen from the Terminal; how to run doxygen as part of Project Builder's build command; and, finally, how to extend doxygen to provide diagrams with an application called dot.

The first step to using doxygen is getting a copy of the program. There is a binary executable available at the doxygen website listed below (version 1.3.4 at the time I finished this article). Download the file, decompress it and place the resulting folder somewhere. I placed mine in the Applications folder.

Welcome to the Shell

Read the following paragraph:

You will need to install the doxygen binary along your shell's path. You can then test your installation by changing to the doxygen/examples/ directory, removing the directory called afterdoc, then executing the command doxygen afterdoc.cfg (which should regenerate the directory). You can browse the generated documentation by opening the file index.html inside the afterdoc/html/ directory.

Got it? If so, you can probably just skim the installation section of this article. If only some (or even none) of the paragraph made sense to you, please keep reading while I try to lead you through the steps carefully in the rest of this section. What I cannot do is give you a full introduction to the UNIX shell (I am still learning it myself). If that is your goal, the most important UNIX command is man. I used it to confirm that I was correct when I described each command below. You have probably read it before in MacTech, but I will write it again: use man frequently as you explore the shell.

Start up Terminal (in Applications/Utilities) and you are now in a shell. If you are reading this section, I will assume that you are working in the tcsh shell without much modification (for example, the default settings from Apple). Below is a description of how to determine if that assumption is accurate. You should type in the command following the % symbol and the computer should respond with something similar to the remaining lines. Your prompt might be longer than just a single % (probably indicating the computer name and your user name), but I removed that information from the examples below. I will use [return] to indicate pressing the return key, and, similarly, [tab] for the tab key.

```
% echo $version[return]
tcsh 6.10.00 (Astron) 2000-11-19 (powerpc-apple-darwin)
options 8b,nls,dl,al,sm,rh,color
```

The command echo asks the shell to print the result to the screen and the expression $version asks for the value of the variable version. If you are not running tcsh, it might seem like the commands chsh and chpass will allow you to change shells. Well, they don't (at least not on my machine). However, the application NetInfo Manager (located in the Applications/Utilities folder) will allow you to change your login shell. I am going to assume that you are using tcsh. You can place the doxygen executable anywhere, but since you will be executing it from a UNIX shell, you will need to make sure the shell can find the executable. The shell keeps a list of directories that it looks through when you issue a command. This information is stored in the variable path. You can discover the directories in this list using the command.

```
% echo $path[return]
/sw/bin /sw/sbin /bin /sbin /usr/bin /usr/sbin
/usr/local/teTeX/bin/powerpc-apple-darwin-current
/usr/local/bin /usr/X11R6/bin
```

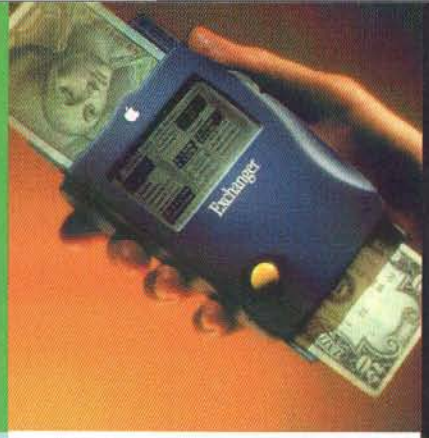
Aaron teaches mathematics at Central Washington University for a living and enjoys mountain biking in the summer and brewing beer in the winter. You can reach him at monsterworks@mac.com.

32

< 1 year

2 years >

$64

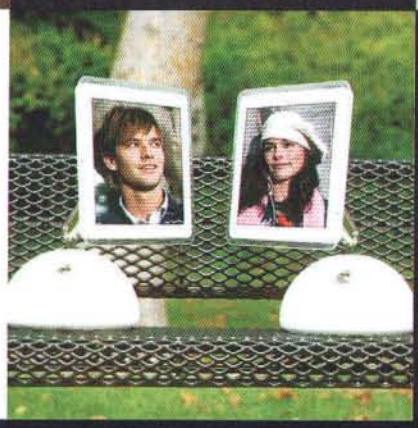


Interviews

Tapping into the world of celebrities and their Macs, only MacDirectory offers exclusive interviews. Get a close and personal view from Sarah Jessica Parker, Sting, Steve Jobs, Madonna, Harry Connick Jr., George Lucas, Jennifer Jason Leigh, Steve Woz and other leaders in the Mac community.

Features

Designers, writers, musicians, business leaders & our technical expert team offer their own personal interpretation of things that only the Mac system can deliver. Featuring over 240 pages of reviews, interviews, news, insights, trends and the largest Macintosh buyers' guide including over 5,000 Mac products and services.



Culture

MacDirectory takes you to the wildest corners of the world and uncovers how Macintosh computers are being used by other cultures. Travel to Japan, Australia, Germany, Brazil, India, Russia & learn more about Apple's cultural impact around the globe.

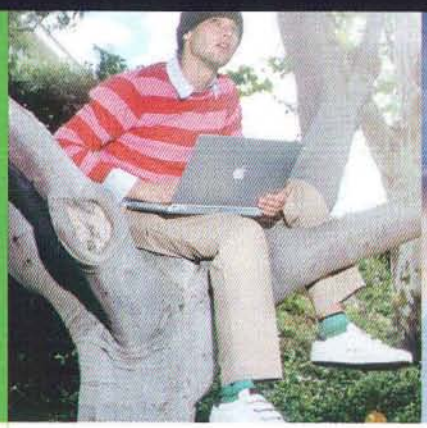
MacDirectory

Beyond any Macintosh Magazine. Subscribe.

< Subscribe

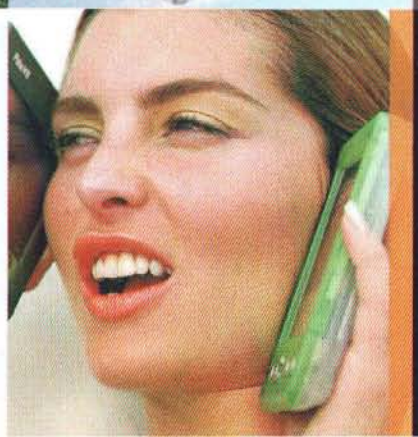
ww.macdirectory.com/mw.html

SEND CHECK OR MONEY ORDER TO:
MacDirectory Sub Dept.
326 A Street, 2C
South Boston, MA 02110



Reviews

Find out all you need to know about the latest Mac products including the hottest Mac OS software and hardware.



Win!

Subscribe to MacDirectory and you will automatically enter our sweepstakes for a chance to win a new Titanium!

If you have not installed anything, your path will be much shorter. Here is some idea of what these directories hold (based on my poking around inside of them): bin/ directories hold binaries (applications), sbin/ directories hold system binaries and usr/ directories hold user files. When I installed teTeX (a TeX installation), the /usr/local/ directories were added. When I installed fink (an application for managing UNIX packages), the /sw/ and /usr/X11R6/ directories were added. We will discuss how to add directories to your path later in the article. For now we will use one of the standard directories that Apple sets up: /usr/bin/. If you decide to add dot following the instructions I provide below, you will create the /usr/local/bin/ directory and may want to keep dot and doxygen together inside this directory (and make appropriate changes below).

You could either move the application into one of these directories using mv or copy it using cp, or do what I am going to describe below and make a link (the UNIX version of an alias). Personally, I place the application in the Applications folder so that it is easy to get to from the Finder and then create a link into the appropriate directory along the path.

There are two types of links under UNIX: hard and symbolic. You will almost never use a hard link. Deleting a hard link causes the original file to be deleted, a very non-Mac behavior (and so not something that should be set up lightly). Symbolic links behave like aliases, deleting the link leaves the original file alone. The system stores a symbolic link as a path to the target so replacing one folder in the path with an identically named folder will change the symbolic link's target. This allows you to install upgrades by replacing folders in the Finder. In order to facilitate this, I removed the version information from the doxygen folder's name.

The following is the command to create a link in the Terminal. You will need to be logged in as an administrator (for example, using the account you created when you first installed Mac OS X). Assuming that you are mimicking my set up, here is the command (type it on one line, it is split here to fit in the article).

```
%ln -s /Applications/doxygen/bin/doxygen
/usr/bin/doxygen[return]
ln: /usr/bin/doxygen: Permission denied
```

As a normal user, you are not allowed to touch the /usr/bin/ directory. The trick is to find a phone booth and change into your super user outfit. This is not as hard as it seems. The command you want to use is the following.

```
%sudo ln -s /Applications/doxygen/bin/doxygen
/usr/bin/doxygen[return]
```

You should be prompted for your password and once it is accepted, the link will be created (assuming you are using an administrator account). Here is a quick breakdown on the command: sudo tells the shell that it should execute this command as an administrator (which is why you will need to type in your password). The command ln creates a link. The option -s creates a symbolic link. The next two arguments are the path to the original file (in Applications) and the path to the alias you are creating. You may need to change this (for example, if you placed doxygen in the Developer folder instead of the Applications folder). If either of these paths contains spaces, you will need to put the path in quotes or escape the space with a \. Once authenticated, you can use sudo for the next 5 minutes without reentering your password. You may also want to make symbolic links for doxysearch and doxytag in case you decide to use them later.

You can test this installation by typing the command below.

```
%which doxygen[return]
/usr/bin/doxygen
```

The which command tells you which file the shell will use when you use the command doxygen. On my computer, the file /usr/bin/doxygen will be used and all is good. But your response may have been the much less satisfying.

```
%which doxygen[return]
doxygen: Command not found.
```

The response here does not look very hopeful. The problem is that the shell only searches the directories in the path variable at startup and when the path variable is changed. In this case, we have added the doxygen command behind the shell's back and need to tell it to recheck the path directories. The command rehash does this.

```
%rehash[return]
%which doxygen[return]
/usr/bin/doxygen
```

Next we will use one of the doxygen examples to make sure doxygen will run. The command cd changes directories. We will use command completion to get to the examples directory. Start by typing the following (there is no [return], this is not a misprint, do not type [return] yet).

```
%cd /Appl[tab]
```

On my machine (and possibly on yours) there are multiple directories starting /Appl (for example, Applications and Applications (Mac OS 9)). Pressing [tab], completes the path to the longest sequence of common characters and you will need to type in some more characters to indicate which one you want. This is tcsh's command completion feature. If you do not remember what the next character should be, you can type [^D] (control-D) to get a list of all the possible completions. As before, if you have spaces in your directory names, you will need to precede them with a \, similarly, parentheses need to be escaped. Now continue typing (without hitting [return] yet).

```
/dox[tab]ex[tab]
```

Assuming your set-up mimics mine, the command should now read

```
%cd /Applications/doxygen/examples/
```

and pressing [return] will execute the command on the line.

You can check the contents of this directory by typing the command

```
%ls | more[return]
```

This ls command asks the shell to list the directory's contents, but rather than just scroll it all onto the screen, we send the output to the more command using a pipe (|). The more command will wait for you to press [space] before scrolling down a page ([return] will scroll down one line and q will exit the listing). Next you want to remove the afterdoc/ directory. The rm command is used to remove (delete) things.

```
%rm afterdoc[return]
rm: afterdoc: is a directory
```

No good, plain rm only works for files. You will need to use the -r option to recursively remove the directory (rmdir -p also works, but is more cautious, requiring the directories to be empty). We are now going to use tcsh's history feature to edit the incorrect command. Press the up arrow and the last command you executed will return to the command line (repeated up arrows will work back through older commands and down arrows work forward), now use the right/left arrow keys to get to the spot just after rm and add a -r between the rm and afterdoc to get the command below (execute it by pressing [return]).

```
%rm -r afterdoc
```

You can confirm that the directory was removed with ls | more (either type it in or use the up arrow a few times to return this to the command line and then press [return]). The file should be in the first page of listings. Watch out with ls, it places all capital letters before all lower case letters, so if you are looking for a file starting with "a", it may actually appear after files starting with "Z". Once you confirm that afterdoc/ is missing, you can type q to exit the listing. Now we are going to use doxygen for the first time.

```
%doxygen afterdoc.cfg | more [return]
```

You might want to scan through the messages or type q to get back to the command line. You can check that this worked with the following.

```
%cd aft[tab]/h[tab][return]
%open index.html[return]
```

The cd command should move you to the afterdoc/html/ directory and the open command opens index.html in your Mac OS X web browser. You could also navigate to this folder in the Finder and open the file with a double click.

The Configuration File

Okay, now it is time to create your own documentation. The first step is to get a template configuration file. When creating the sample project I created a folder called doxygen in the same folder as the project file in the Finder. I then moved to the new directory (with a cd command) and then typed.

```
%doxygen[return]
```

Like many UNIX commands, using the command doxygen with the wrong number of arguments provides a brief statement describing how to use the command. Unless you actually have a file called Doxyfile in the current directory, you will get a list of uses for doxygen. I created a default template file (using the -g option).

```
%doxyfile -g Fish.doxycfg[return]
```

I do not use the default name Doxyfile because I like to be able to tell the associated project from the configuration file's name and by not naming my file Doxyfile, I can use the doxygen command to get help. We now need to adjust the file to suit our needs. You can edit the file in a Mac OS X text editor (such as Project Builder, AlphaX or BBEdit) or a UNIX text editor (such as emacs, vi or pico). Trying to describe which choice you should make is a can of worms I will not even try to sort out here. For those who are new to UNIX, I would suggest one of the Mac OS X editors listed above. You will be able to open the file from the Finder with a double-click (once you set its **Open with...** field in the **Get Info** dialog) or from within the **Open** dialog of the application. Alternatively, you can use the command below to open the file with Project Builder from the shell (assuming Project Builder is in the location Apple placed it). The -a option allows you to specify the application that will be used to open the file. If you find that you are doing a lot of opening with the same application, check out the section below where I talk about the .tcshrc file.

```
%open -a /Developer/Applications/Project\ Builder.app
Fish.doxycfg[return]
```

The template file is heavily commented (unless you used the -s option when building it) and is also described in the doxygen documentation. I only adjusted the lines shown below for the sample project.

```
PROJECT_NAME   = "About Box"
OUTPUT_DIRECTORY = ./doxygen
EXTRACT_ALL = YES
EXTRACT_PRIVATE = YES
EXTRACT_STATIC = YES
TAB_SIZE = 4
OPTIMIZE_OUTPUT_FOR_C  = YES
INPUT = ./Source
SOURCE_BROWSER = YES
ALPHABETICAL_INDEX = YES
GENERATE_LATEX = NO
```

The PROJECT_NAME line sets up a project name that will be used in the documentation and the TAB_SIZE line indicates that I use tabs that are four characters wide (since doxygen will convert

tabs to spaces when formatting the documentation). I also told doxygen that all source is C source (in the OPTIMIZE_OUTPUT_FOR_C line). The SOURCE_BROWSER line tells doxygen to include the source of the files as part of the documentation. I also requested an alphabetical index of all of the compound structures used with the ALPHABETICAL_INDEX line.

The OUTPUT_DIRECTORY line specifies where the output is to be placed. The directory called dot (.) refers to the current directory and dot dot (..) is the parent directory of dot (these are available in the shell so you can use them there also). To prepare for later work, I will assume that the current directory is the directory containing the project file (and the parent of the directory containing the configuration file). If you plan to execute the doxygen command from same directory as the configuration file, then you might want to use dot as the output directory. The GENERATE_LATEX line turns off the option to generate LaTeX documentation (the template file creates both HTML and LaTeX documentation, rtf and man documentation are available also but turned off in the template). The INPUT_DIRECTORY specifies where the header and source files reside. If you were planning to execute from within the same directory as the configuration file, you will need to change the input directory to ../Source.

The three lines concerning EXTRACTion are telling doxygen that I want everything documented. If I wanted to hide implementation details, I could set these to NO and limit the visibility of the implementations in the documentation. Toggling these provide you with control over what is documented and allow you to create multiple sets of documentation, for example, one set for people developing the package (with these set to YES) and another for people who are using the package (with these set to NO and maybe even the SOURCE_BROWSER also set to NO). If you want more details about what will be extracted, please consult the doxygen manual, however, there is one detail worth repeating here. If you have EXTRACT_ALL set to NO, definitions within namespaces are only documented if the namespace is documented. Similarly, global definitions will only be documented if the file in which they occur is documented (with a @file tag).

A good way to figure out what doxygen can do is to poke around in the configuration file and to toggle some switches and compare the output. Although the configuration file is long (over 1000 lines including comments), the comments help guide you through the various options. You can test the configuration file from the command line by first changing to the directory containing the project file and then issuing the command

```
%doxygen doxygen/Fish.doxycfg[return]
```

This should generate an html folder inside the doxygen folder full of documentation that you can view with your browser.

Preparing to Doxygenate Your Code

You can run doxygen without using any special comments, however, you get better results by adding them. This section is short because I found the manual and examples included with doxygen to be quite good and suspect that the primary obstacle to using it on Mac OS X is going to be the installation process described above. While reading the examples here (and those provided with doxygen), be aware that doxygen is configurable, and if the comment style below feels unnatural to you, be sure to read the doxygen manual to see if your preferred comment style is supported.

To prepare your code so that doxygen can generate good documentation you will need to place special comment blocks inside your code. There are a wide variety of options using both C and C++ style comments. I tend to use C++ style comments (so that I can use C style comments to eliminate long blocks of code during testing). If you prefer C style comments, please refer to the doxygen manual for the ways they can be used (alone or in conjunction with C++ style comments). When reading a source file, doxygen will recognize comments that begin with either /// or //! as a special comment that contains documentation information. I prefer to use //! because it sticks out when I visually scan a file. You can structure the comments with a limited subset of HTML and by doxygen tags (you can use either @ or \ as the escape character, I use @ because it sticks out when I visually scan a file). I have created a (somewhat silly) project to give you some idea of what doxygen can do. The excerpt below is intended to provide you with a feel for the type of comments that can be added to the source to provide documentation. As usual in magazine printouts of code, watch out for wrapped lines (or better yet, look at the actual source file, available at the MacTech site as well as my website listed below).

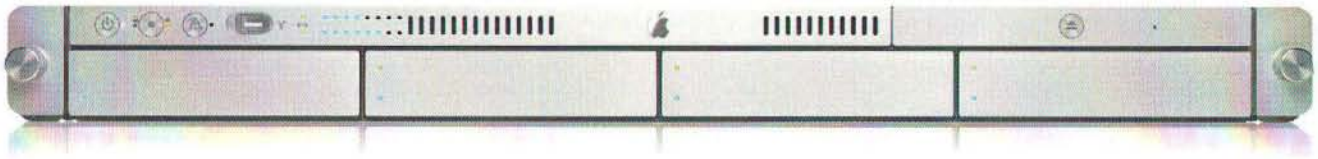
excerpt from fish.h

```
//! @brief This structure information about fish in my freezer.
//!
//! It doesn't really do justice to the great variety of things that are fish
//! in my freezer. For instance, it might be nice to know how old they are or
//! whether they still have their heads.
typedef struct
{
  fishTypes myType; //!<what type of fish we are describing
  int myWeight;     //!<the weight of the fish
} fish_t;

//! @brief The number of fish that fit in my freezer.
//!
//! Of course, it really depends on the size of the fish and if
//! I upgrade my freezer I will certainly need to enlarge this number.
//! The only reason for all this typing is create a detailed description.
#define MAX_FISH 10

//! @brief The fish in my freezer.
extern fish_t freezer[MAX_FISH];

//! @brief Fills the freezer with fish.
void FillFreezer();

//! @brief Returns the total weight of all fish of one type in the freezer.
int Weigh(fishTypes inType);
```

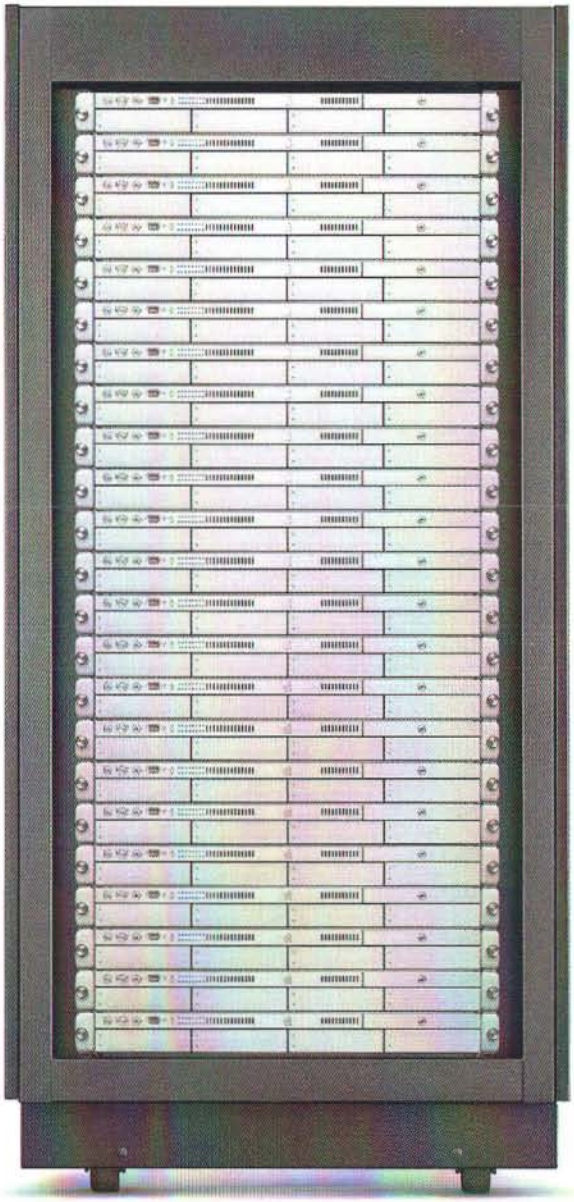
excerpt from fish.c

```
//! This function iterates through all the fish in my freezer of
//! a particular type and sums up their weights.
//! This function replaces the defective WeighEm() function.
//! @param inType is the type of fish I want to weigh.
//! @returns the total weight of all fish of that type in my freezer.
//! @bug I can't figure out what's going wrong with this function.
int Weigh(fishTypes inType)
```

Xserve



Density optimized rack mounted Mac OS X Server



UNIX-based Server Solutions from Apple

Nearly half a terabyte of storage per 1U

630 gigaflops of processing power

Hot-Swappable drives

Industry-standard 1U rack-optimized design

There has never been a better time to buy...

Small Dog Electronics carries factory refurbished models too offered at substantial savings!

(subject to availability)

Small Dog Electronics

www.smalldog.com

1673 Main Street, Route 100, Waitsfield, Vermont

Apple Specialist

To learn more: http://www.smalldog.com/xserve/

```
{
  int i, result;
  for (i = 0; i < MAX_FISH; i++)
  {
    if (freezer[i].myType == inType)
    {
      result = result + freezer[i].myWeight;
    }
  }

  return result;
}
```

When processing the code, doxygen will look for documentation for #defines, enums, structs, global variables and functions (and other things) and will produce both a brief description as well as a detailed description. As a convention, I use the @brief tag for brief comments although doxygen has a default algorithm for determining the brief and detailed descriptions if you do not specify them. In the excerpt above, @param indicates a parameter to the function, @returns describes the return value and a @bug tag adds this comment to the bug list (bonus question: can you find the bug?). If the comment contains something that looks like a function call (for example, the WeighEm() above), doxygen will try to hyperlink it to the appropriate function's documentation. Another nice feature is that doxygen will try to crosslink everything to anything referred to by it and anything to which it refers. The following is a text rendition of the documentation for the function Weigh(). I have indicated the hyperlinks with underlines.

Sample Documentation

```
int Weigh
(
  fishTypes
  inType
)

Returns the total weight of all fish of one type in the
freezer.

This function iterates through all the fish in my freezer of
a particular type and sums up their weights. This function
replaces the defective WeighEm() function.

Parameters:
  inType
    is the type of fish I want to weigh.

Returns:
  the total weight of all fish of that type in my freezer.

Bug:
  I can't figure out what's going wrong with this function.

Definition at line 38 of file fish.c.

References freezer, MAX_FISH, fish_t::myType, and
fish_t::myWeight.

Referenced by main().
```

You can find more samples in the doxygen examples as well as the project associated with this article (at the MacTech site as well as my website listed below).

Adding A Doxygen Stage to Your Build

I am going to leave further exploration of what doxygen does to the doxygen manual and examples and move on to controlling doxygen from within Project Builder. I will describe how I added a step to create the documentation at the end of the build. You could place this step at other points in the build process or even create a special target that generates documentation and nothing else. I selected the **Edit Active Target 'Fish'** menu item from the **Project** menu. In the left pane, I expanded **Build Phases** (well, not quite, it was already expanded), and added a new build phase here by selecting the **ResourceManager Resources** item and then **New Shell Script Build Phase** in the **New Build Phase** submenu of the **Project** menu. There are two places I needed to add information. The first is the **Shell:** text field. This field is set to /bin/sh (the Bourne shell) by default. The Bourne shell is often used for scripts for portability reasons, but I want to use tcsh for this project since our script is a simple single line command and I spend most of my time in tcsh. To do this, I changed the /bin/sh to /bin/tcsh. The second field is the command to be executed and in this case I typed doxygen doxygen/Fish.doxycfg. That wasn't so tough. Now at the end of any successful build, doxygen will create a folder full of documentation (and doxygen's messages are sent to the Build window). If you just have a couple of simple commands to execute in the shell during a build, you can add them with separate Shell Script Build Phases. For more complicated steps, you might want to actually write a file that contains the script and then use the source command to execute that file from Project Builder.

Doxygen and Dot

When creating documentation, doxygen is also able to generate diagrams describing various relationships — in particular, header inclusions and class hierarchies. In order to do this, doxygen needs to use an application called dot. First download dot for Mac OS X from the GraphViz website listed below in the references (my copy was version 1.10 built on 08/27/2003). If working with the shell is still relatively new, you should grab the Installer package (which is what the instructions below assume because that is what I did). Download it and install it according to the instructions. I would suggest letting it place everything in the default location and the instructions below assume that you did this. If you place it somewhere else, you will need to do a few more steps (information can be found at the GraphViz website). You could check to see if the files were installed with the following commands.

```
%cd /usr/local/bin[return]
%ls dot[return]
dot
```

When I asked the shell to ls dot, I am asking it to list every file called dot and if dot does not exist you will get an error message. You can use * as a wildcard character, for example

```
%ls *dot*[return]
dot    dot2gxl    dotneato-config    dotty    gxl2dot
```

This is more convenient than using ls | more in places where you know some part of the filename. Now that you know the file exists, find out if the shell can find the installation with which.

```
%which dot[return]
/usr/local/bin/dot
```

In my case, everything is good. However, just like with doxygen, it is likely that your experience was not quite so satisfying.

```
%which dot[return]
dot: Command not found.
```

The rehash command probably will not solve this problem. The problem is that the shell (as set up by Apple) does not know that it should be searching in /usr/local/bin/ for commands. We need to add some directories to the variable path. At startup, tcsh will read a number of files (/etc/csh.cshrc, /etc/csh.login and ~/.tcshrc, the ~ refers to your home directory). If you are interested in exactly when they are read, you should man tcsh. A good place to fiddle with your path is in ~/.tcshrc. I am going to suggest you use pico to edit the file since this is a simple editing job. More complicated files should probably be handled with more complicated editors (like emacs, vi, AlphaX or BBEdit). Notice that cd is the same as cd ~ (it will save you two characters of typing, which over your lifetime... well, probably still won't amount to too much).

```
%cd[return]
%pico .tcshrc[return]
```

The pico editor is very simple and even provides you with two lines of commands at the bottom of the screen. Here are some of the important lines that occur either in my /etc/csh.login or my ~/.tcshrc files (watch the wrapping, there are three lines here, they start with set, setenv, alias).

```
set path = ( $[path] /usr/local/bin )
setenv DYLD_LIBRARY_PATH /usr/local/lib/graphviz
alias alpha "open -a /Applications/Alpha/AlphaX.app"
```

The set line will add the /usr/local/bin/ directory at the end of the search path for files, once this is added, the tcsh should be able to find dot. In order to run, dot needs to determine where its dynamically loaded libraries are placed. The setenv sets up the information in an environment variable so that dot will be able to find this information while it is running. The next line show how to create aliases that can be used to open files in applications from the Terminal. In my case, the command alpha is equivalent to open -a /Applications/Alpha/AlphaX.app. To open .tcshrc in AlphaX, I can type the following command.

```
%alpha ~/.tcshrc
```

Unfortunately, the open command does not create a new file. If you want to create a file and then open it, one trick is to first touch the file first (touch updates the last modification date of the file and if the file does not already exist, it will create the file). You can use a semicolon to place two commands on a single line.

```
%touch blah; alpha blah
```

You need to add the first two lines to your .tcshrc file (the other line was provided because it may be useful). Once you type these lines, save the document by typing [^O] (control O, write Out) and hit [return] to accept the filename. Then exit pico by typing [^X] (control X, eXit). You can cause the commands in this file to be executed by using the source command.

```
%source .tcshrc
```

Now, when you ask which dot, the shell should find it (you changed the path variable so the shell automatically executed rehash for you).

```
%which dot
/usr/local/bin/dot
```

Reopen the Fish project and change the command in the Shell Script Build Phase to doxygen doxygen/Fish2.doxycfg (notice the 2). The only difference between Fish.doxycfg and Fish2.doxycfg is that Fish2.doxycfg has specified that we have dot (HAVE_DOT = YES) and that we will be using gif style images (DOT_IMAGE_FORMAT = gif). Now rebuild the project and then browse the new documentation.

Conclusion & Bibliography

The steps above have focused on getting doxygen installed, but many of the bits and pieces are common techniques for installing programs on UNIX systems. Although I have not had a chance to work with Xcode, I suspect that the experience will be very similar to Project Builder. Hopefully you will take the time to try working with doxygen as well as finding other Mac OS X command line applications. If you find yourself using doxygen, please consider using the PayPal link from the doxygen website to show your appreciation. If you are really ambitious, doxygen can be extended to work with more languages than its current ones (it is open source). For example, an ambitious programmer could write code to support Objective C.

Here are the sources I used to put together this article as well as URLs mentioned in the article.

- doxygen web site : www.doxygen.org/
- GraphViz (dot) web site: www.phil.uu.nl/~js/graphviz/
- introduction to tcsh: **Using csh & tcsh** by Paul DuBois (O'Reilly & Associates, Inc.)
- my web site: www.cwu.edu/~montgoaa/
- MacTech web site: www.mactech.com/

by Tim Monroe

Strange Brew

Running QuickTime for Java Applications on Windows

Introduction

In the two previous *QuickTime Toolkit* articles (in *MacTech*, January and February 2004), we took a look at using Apple's QuickTime for Java classes to build a simple Java-based Macintosh application that can manipulate QuickTime movies. Our application — called "JaVeez" — can open movie files and display their movies in windows on the screen. **Figure 1** shows a typical movie window displayed by JaVeez.

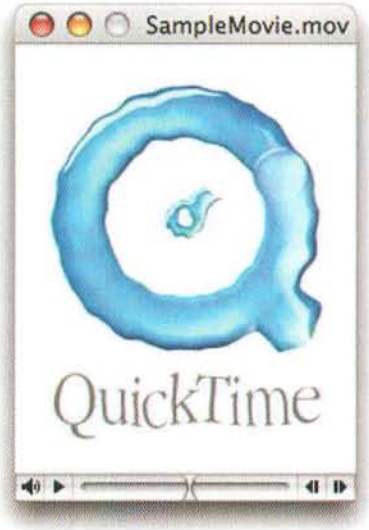


Figure 1: *A JaVeez movie window (Macintosh)*

JaVeez supports all the standard editing operations on linear movies and can save edited movies into their original files or into new files. If the user tries to close an edited movie file, JaVeez displays the dialog box shown in **Figure 2**, which gives the user the opportunity to save or discard those edits.

Figure 2: *The Save Changes dialog box (Macintosh)*

In this article, I want to investigate what needs to be done to get JaVeez to run on Windows operating systems as well as on Mac OS X. In other words, I want to see how to turn the window in **Figure 1** into the window in **Figure 3.**

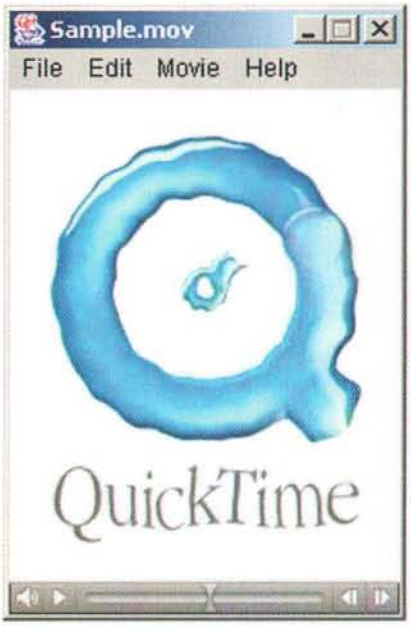


Figure 3: *A JaVeez movie window (Windows)*

Tim Monroe is a member of the QuickTime engineering team at Apple. You can contact him at monroe@mactech.com. The views expressed here are not necessarily shared by his employer.

And I want to see how to turn the dialog box in **Figure 2** into the similar dialog box in **Figure 4**.

Figure 4: *The Save Changes dialog box (Windows)*

The good news is that we don't really need to do very much work at all to get JaVeez running on Windows, thanks to its Java underpinnings and to the platform-independent graphical toolkits AWT and Swing. We need to add a small amount of platform-specific code to adjust the placement of the Quit and "About JaVeez" menu items, and we need to make a few adjustments to our Xcode project. But ultimately, and indeed fairly quickly, we'll have a single executable file that can be launched on both Mac OS X and Windows computers.

Once we've got JaVeez running on Windows, we'll take a look at extending it on both platforms by allowing the user to resize its movie windows. To do that, we'll need to install a movie controller action filter procedure. As we've had some trouble doing that with a few of the development environments we've considered in recent articles, it will be interesting to see how easy it is to do this using QuickTime for Java.

Menus

Let's begin by considering the code changes we need to make to get JaVeez running properly on Windows, all of which are related to the creation and handling of menus. On Mac OS X, the "About JaVeez" and Quit menu items are contained in the Application menu. On Windows, those items need to be moved into the File menu and the Help menu, respectively. **Figure 5** shows the File menu on Windows. Notice that the Quit item is labeled "Exit" on Windows.

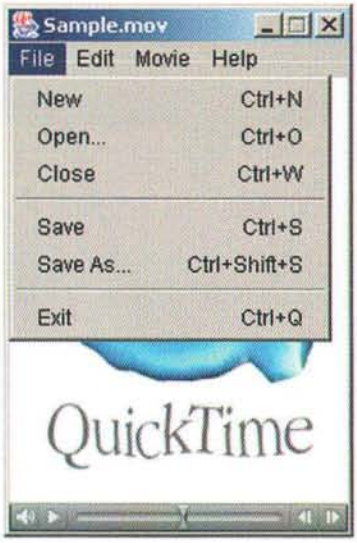


Figure 5: *The File menu of JaVeez (Windows)*

Figure 6 shows the Help menu on Windows.

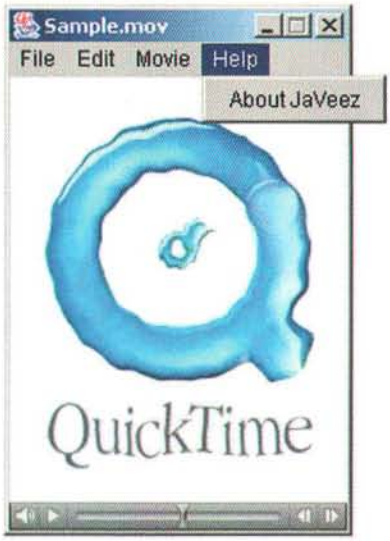


Figure 6: *The Help menu of JaVeez (Windows)*

First we need to create the new menu and menu items; then we need to attach actions to those new menu items.

Creating Menus and Menu Items

It's easy enough to add a separator line and the Exit menu item to our existing File menu. The only mildly interesting hoop we need to jump through is to figure out whether JaVeez is running on Macintosh or Windows. There are in fact at least three ways we can do this.

The standard way to check whether a Java application is running on Macintosh is to look at the mrj.version property, like this:

```
if (System.getProperty("mrj.version") != null)
```

"MRJ" stands for "Macintosh Runtime for Java", which was the official name of earlier versions of the Java support on Macintosh computers. If the mrj.version property exists, then the string object returned by the System.getProperty method is non-null and hence our application is running on some version of Mac OS.

While this way of determining that our application is running on Mac OS is still supported in JS2E version 1.4.1 and later, it is no longer the recommended way of doing so. A better way is to look at the os.name property, like this:

```
if (System.getProperty("os.name").equalsIgnoreCase(
                                        "Mac OS X"))
```

This expression evaluates to true just in case our application is running on some version of Mac OS X.

There is yet a third way to determine the current system our application is running on, using the QTSession class we encountered in the two previous articles:

```
if (QTSession.isCurrentOS(QTSession.kMacOSX))
```

We'll use this third way throughout our application. For instance, **Listing 1** shows the code we'll add to the addFileMenuItems method to add the Exit menu item to the bottom of the File menu.

Listing 1: Adding items to the File menu

addFileMenuItems

```
if (!(QTSession.isCurrentOS(QTSession.kMacOSX))) {
  fileMenu.addSeparator();
  miExit = new MenuItem(resBundle.getString("exitItem"));
  miExit.setShortcut(new MenuShortcut(KeyEvent.VK_Q,
                                      false));
  fileMenu.add(miExit).setEnabled(true);
  miExit.addActionListener(exitAction);
}
```

And **Listing 2** shows the complete revised version of the addMenus method.

Listing 2: Configuring the menu bar

addMenus

```
public void addMenus () {
  editMenu = new Menu(resBundle.getString("editMenu"));
  fileMenu = new Menu(resBundle.getString("fileMenu"));
  movieMenu = new Menu(resBundle.getString("movieMenu"));

  addFileMenuItems();
  addEditMenuItems();
  addMovieMenuItems();

  if (!QTSession.isCurrentOS(QTSession.kMacOSX)) {
    helpMenu = new Menu(resBundle.getString("helpMenu"));
    addHelpMenuItems();
  }

  setMenuBar(mainMenuBar);
}
```

The addHelpMenuItems method, which is called by this new version of addMenus, is defined in **Listing 3**.

Listing 3: Adding items to the Help menu

addHelpMenuItems

```
public void addHelpMenuItems () {
  miAbout = new MenuItem(resBundle.getString("aboutItem"));
  helpMenu.add(miAbout).setEnabled(true);
  miAbout.addActionListener(aboutAction);

  mainMenuBar.add(helpMenu);
}
```

Creating Actions

All that remains is to define two new action classes and to create instances of those two classes (which, as we saw in **Listings 1** and **3**, we add as listeners to the Exit and "About JaVeez" menu items). **Listing 4** shows how we can handle the Exit menu item on Windows. As you can see, we simply call the existing method attempQuit, which inspects all open movie windows and gives the user the chance to save or discard any unsaved changes.

Listing 4: Handling the Exit menu item (Windows)

ExitActionClass

```
public class ExitActionClass extends AbstractAction {
  public ExitActionClass (String text, KeyStroke shortcut) {
    super(text);
    putValue(ACCELERATOR_KEY, shortcut);
  }
  public void actionPerformed (ActionEvent e) {
    attemptQuit();
  }
}
```

Similarly, **Listing 5** shows how we can handle the "About JaVeez" menu item on Windows; again, we just call an existing method, about.

Listing 5: Handling the About menu item (Windows)

AboutActionClass

```
public class AboutActionClass extends AbstractAction {
  public AboutActionClass (String text) {
    super(text);
  }
  public void actionPerformed (ActionEvent e) {
    about();
  }
}
```

Finally, we need to add a few lines of code to the createActions method to create instances of these two classes; **Listing 6** shows the code we need to add.

Listing 6: Creating actions

createActions

```
if (!(QTSession.isCurrentOS(QTSession.kMacOSX))) {
  exitAction = new ExitActionClass(
                resBundle.getString("exitItem"),
                KeyStroke.getKeyStroke(
                        KeyEvent.VK_Q, shortcutKeyMask));
  aboutAction = new AboutActionClass(
                resBundle.getString("aboutItem"));
}
```

When this is all done, the File and Help menus will have the appearance shown earlier in **Figures 5** and **6**. Selecting the Exit menu item performs the standard application shutdown operations. And selecting the "About JaVeez" item displays the dialog box shown in **Figure 7**.

Figure 7: *The About box of JaVeez (Windows)*

Note that we haven't changed any of the code called by the new actions. Our About box looks pretty much the same as its Macintosh counterpart, thanks to the wonders of Java and Swing.

Java Applications

Recall that we started building JaVeez by creating a new Xcode project based on the "Java AWT Application" template. The target built by Xcode is a double-clickable Macintosh application named JaVeez.app. This application responds appropriately to the basic Apple events OpenApplication, OpenDocuments, and QuitApplication. This means, in particular, that JaVeez automatically opens QuickTime movie files and Flash files dropped onto its icon in the Finder or in the Dock.

In fact JaVeez.app is just a directory that holds some special items. We can inspect those items by option-clicking on the JaVeez icon and selecting the "Show Package Contents" item. A Finder window opens that holds a folder named Contents. The Contents folder contains a folder named Resources, which in turn contains a folder named Java. **Figure 8** shows the contents of the Java folder:

Figure 8: *The JaVeez.jar file*

JaVeez.jar is a *Java archive* file, also called a *jar* file. It contains the executable code of the application along with any images or other resources used by the application. Jar files are the standard distribution containers for Java applications. In theory, we should

be able to move this jar file to a Windows machine, double-click it, and have JaVeez launch and run. In practice, we need to make a couple of minor tweaks to our project for this to work correctly.

Fixing the Manifest

Indeed, double-clicking this existing JaVeez.jar file won't even launch JaVeez on Mac OS X. If we try it, we'll see the warning shown in **Figure 9**.

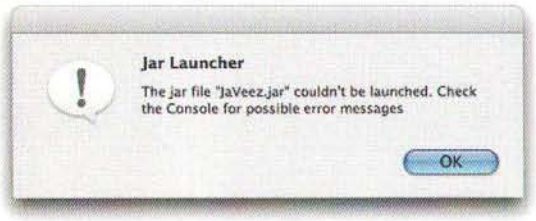


Figure 9: *The launch warning*

The Console log contains this message:

```
Failed to load Main-Class manifest attribute from
/Users/monroe/JaVeez/build/JaVeez.app/Contents/Resources/Java/
JaVeez.jar
```

The problem is that the default jar file's *manifest* (a special file in the archive that contains information about the other files in the archive) does not indicate which class in the jar file is the main class. To fix this, let's add to the project a new file called JaVeez.mf that contains these lines:

```
Main-Class: JaVeez
```

(There is a newline character at the end of the first line; it needs to be there in order for our new manifest to work correctly.) Then we can set this file as the project's manifest file by specifying its name in the target settings panel, as shown in **Figure 10**. When we build JaVeez, the lines in the file JaVeez.mf will be appended to the default manifest data.

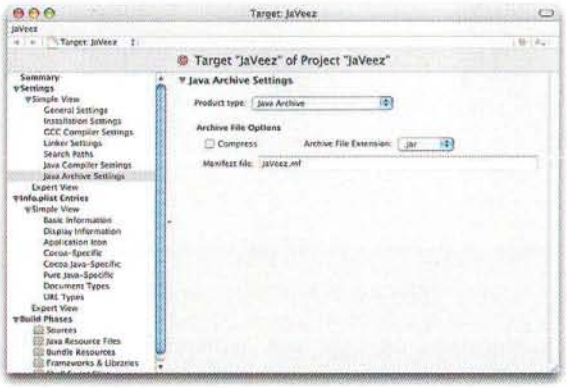


Figure 10: *The manifest file setting*

While we're here, let's adjust one other project setting, to prevent an annoying but harmless warning from being issued each time we launch JaVeez. Select the "Cocoa Java-Specific" pane and *un*check the "Needs Java" box, which is shown in **Figure 11.**

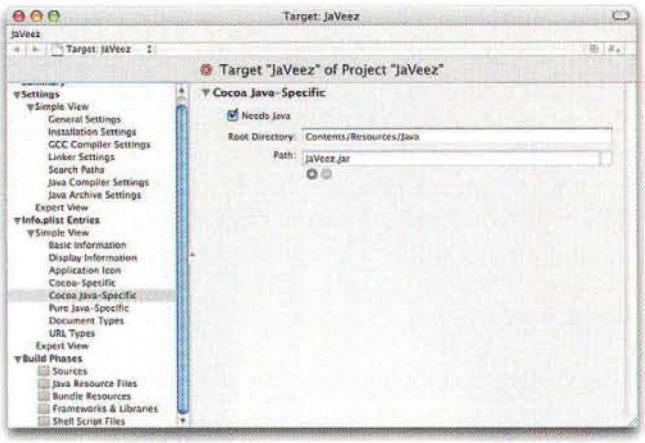


Figure 11: *The "Needs Java" setting*

Leaving that setting at its default value will result in this warning appearing on the console whenever we launch JaVeez:

```
-[NSJavaVirtualMachine initWithClasspath:] cannot
                instantiate a Java virtual machine:
```

Let's rebuild JaVeez so that these changes take effect. Now double-clicking the jar file will launch the application, as desired. That's very cool. What's perhaps not so cool is that when we launch JaVeez by double-clicking the JaVeez.jar file, we lose the Finder interaction we got when we launched it by double-clicking the JaVeez.app file; in particular, we can't drop files onto the JaVeez.jar file or into its Dock icon.

Adding Stub Classes

Can we now move the jar file to a Windows machine and run it? Not quite, but almost. Recall that JaVeez uses the Application class in the com.apple.eawt package to handle the basic application-related Apple events. This package is not implemented on Windows, so we need to do a little bit of work to keep the Windows Java runtime engine happy.

There are several ways to work around this issue. The easiest way is to add some stub classes to our project that provide empty implementations of the classes and methods in com.apple.eawt. Fortunately, Apple provides a set of stubs that can be used for this purpose. Look for the sample code package called AppleJavaExtensions on the Apple developer site. This package consists of a file called AppleJavaExtensions.jar. We need to unpack that jar file; in a Terminal window, execute this command:

```
jar -xf AppleJavaExtensions.jar
```

When this command completes successfully, you'll see two new directories named META-INF and com. Copy the com directory into the JaVeez folder that contains the application project files and source code. Add that folder to the project; the project file now looks like **Figure 12.**

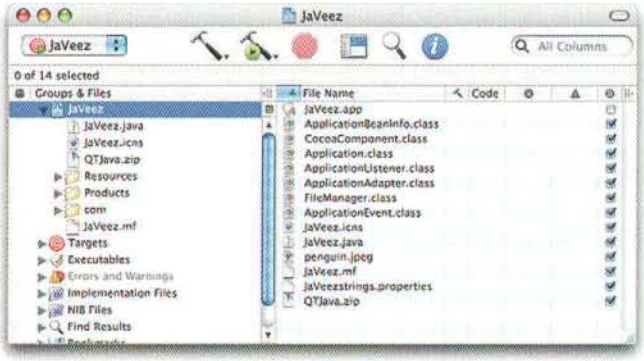


Figure 12: *The updated project file*

Build the project and copy the JaVeez.jar file to a computer running Windows that has Java and QuickTime installed on it. Double-click the jar file and behold the new empty movie window that appears (**Figure 13**).

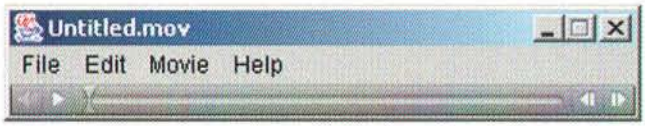


Figure 13: *An empty movie window (Windows)*

Use the Open menu item in the File menu to open a QuickTime movie file and then spend a few minutes verifying that the movie and its window behave as expected. (See **Figure 3** again.)

Let's pause to consider our progress so far. We now have a fully-functional Windows application, and we did that with just a couple of easy adjustments to our existing project and source code. We added some code to handle the different locations of the Quit/Exit and About menu items on Mac and Windows, and we tweaked the jar file's manifest to specify the application's main class. Also, we added a few stub classes to our project to provide empty implementations of the com.apple.eawt package for Windows. Everything runs fine. We're done, right?

Using Reflection

Wrong. To be honest, I'd be very happy to consider our porting work done and then move on to add some features to JaVeez that show off the interesting capabilities of QuickTime for Java. After all, we *do* now have a single jar file that runs on both Mac OS X and Windows and that has all the features we originally planned. But in fact we're not quite finished. The

reason is that the stub-class solution described above, though technically impeccable in terms of getting the job done, has a decidedly non-Java flavor. The generally-preferred solution to the problem of classes being implemented on one platform but not on another is to use a feature of the Java language known as *reflection*. If you're happy using the stub classes, skip to the next section. Otherwise, read on....

Here's the crux of the issue: JaVeez currently contains the following line of code, which creates an instance of the com.apple.eawt.ApplicationAdapter class and attaches it to fApplication.

```
fApplication.addApplicationListener(
            new com.apple.eawt.ApplicationAdapter() {
```

When the Java runtime engine loads a package, it looks for all classes referenced in the package. Even though this line of code won't actually be executed on Windows, the runtime loader still wants to find some code for the ApplicationAdapter class.

Rather than solve this problem by providing no-op implementations of the classes in com.apple.eawt, which is what the stub classes provide, we can factor the relevant code into a new package and then load that package only if JaVeez is running on Mac OS X. The Windows runtime engine never sees the references to com.apple.eawt, so the stubs are no longer necessary. Very nice.

Let's begin by defining a new package, called "osxadapter", which imports the com.apple.eawt classes and the JaVeez application classes:

```
package osxadapter;

import com.apple.eawt.*;
import javeez.*;
```

The osxadapter package contains a single class, OSXAdapter, which defines three public methods: handleAbout, handleQuit, and registerMacOSXApplication. (The code upon which this class is based defines two additional methods for handling an application's preferences; since JaVeez does not support user-selectable preferences, I've taken the liberty of removing those methods.) **Listing 7** shows the complete definition of the OSXAdapter class.

Listing 7: Handling Application menu items

OSXAdapter

```
public class OSXAdapter extends ApplicationAdapter {
  private static OSXAdapter  theAdapter;
  private static com.apple.eawt.Application
                                        theApplication;

  private JaVeez  mainApp;

  private OSXAdapter (JaVeez inApp) {
    mainApp = inApp;
  }

  // handle the About menu item
  public void handleAbout (ApplicationEvent ae) {
    if (mainApp != null) {
      ae.setHandled(true);
      mainApp.about();
    } else {
      throw new IllegalStateException("handleAbout: JaVeez
instance detached from listener");
    }
  }

  // handle the Quit menu item
  public void handleQuit (ApplicationEvent ae) {
    if (mainApp != null) {
      ae.setHandled(false);
      mainApp.attemptQuit();
    } else {
      throw new IllegalStateException("handleQuit: JaVeez
instance detached from listener");
    }
  }

  public static void registerMacOSXApplication (JaVeez inApp) {
    if (theApplication == null) {
      theApplication = new com.apple.eawt.Application();
    }

    if (theAdapter == null) {
      theAdapter = new OSXAdapter(inApp);
    }

    theApplication.addApplicationListener(theAdapter);
  }
}
```

The first two methods, handleAbout and handleQuit, simply call the existing about and attemptQuit methods in JaVeez. They also both call the setHandled method of the ApplicationEvent class to indicate whether or not the event has been handled. Notice that handleQuit needs to pass false to setHandled since we want to allow the user to cancel the application shut-down operation.

The interesting method in the OSXAdapter class is registerMacOSXApplication, which creates an instance of the Application class, creates an instance of the OSXAdapter class, and then adds the adapter object as a listener for the Application object. The registerMacOSXApplication method is the only method that JaVeez needs to call explicitly at run-time. Of course JaVeez cannot invoke that method by name (since it references classes that do not exist on Windows machines); rather, it invokes that method indirectly, using Java's reflection APIs. **Listing 8** shows the definition of the macOSXRegistration method that we need to add to JaVeez.

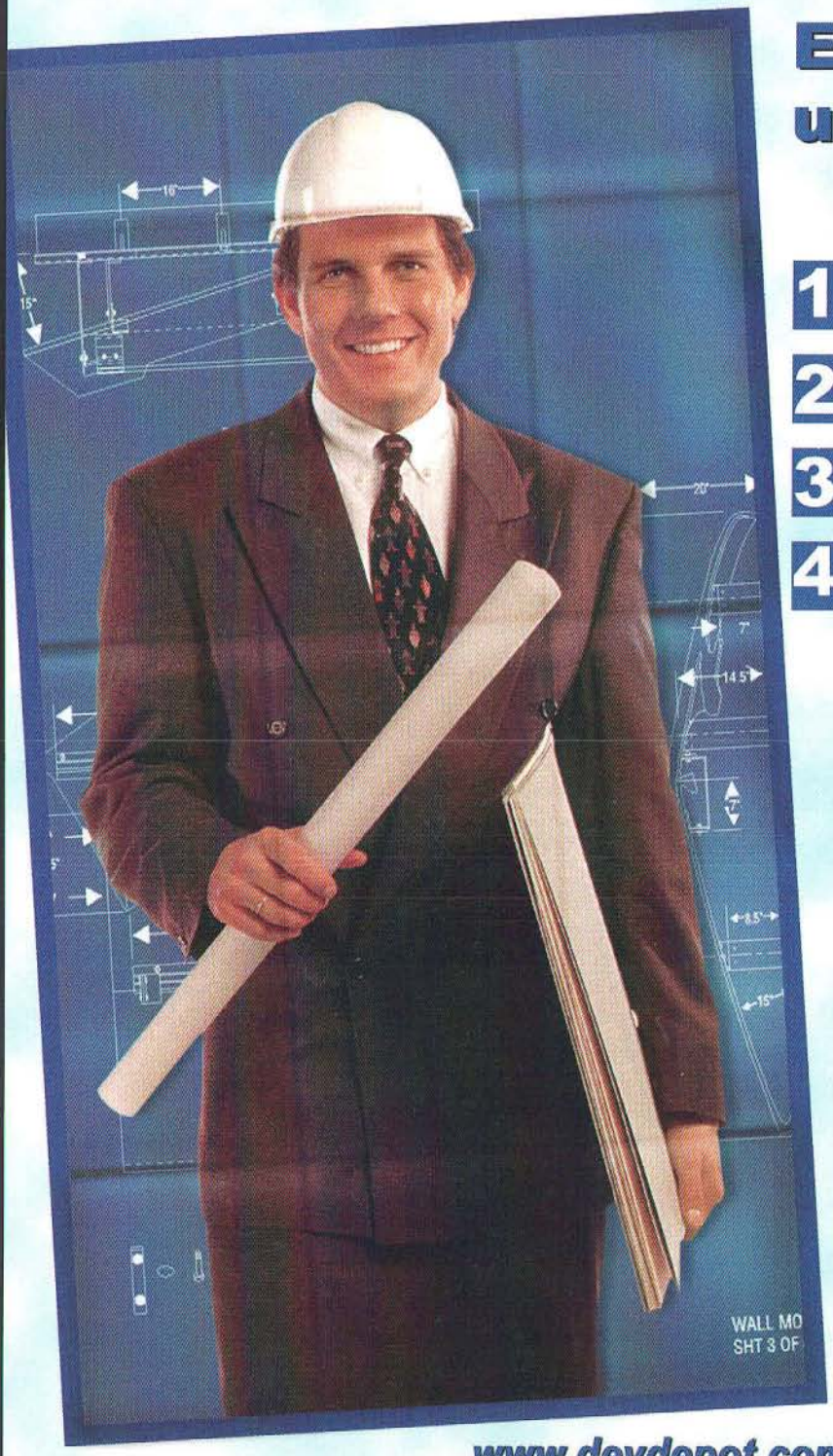
Listing 8: Registering a Mac OS X application

macOSXRegistration

```
public void macOSXRegistration () {
  if (QTSession.isCurrentOS(QTSession.kMacOSX)) {
    try {
      Class osxAdapter =
            Class.forName("osxadapter.OSXAdapter");

      Class[] defArgs = {JaVeez.class};
      Method registerMethod =
                osxAdapter.getDeclaredMethod(
                    "registerMacOSXApplication", defArgs);
      if (registerMethod != null) {
        Object[] args = {this};
            registerMethod.invoke(osxAdapter, args);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

As you can see, macOSXRegistration calls Class.forName to find the class named "OSXAdapter" in the package osxadapter. If it

Ready to Build a Better MAC?



Easy to Install ugrades help you...

1. Add more *HD* space
2. Add a faster *CPU*
3. Add more *RAM*
4. Add new *ports*

We know Macintosh Technology.

Come to *DevDepot* for all the cool upgrades!

DEV DEPOT®

877-DEPOT-NOW

www.devdepot.com/bettermac

finds it, macOSXRegistration executes that method by calling the invoke method.

Callback Procedures

So JaVeez now runs on both Macintosh and Windows computers. We can open one or more QuickTime movies, play them back or interact with them, edit them, and save or discard those edits as desired. JaVeez currently meets or exceeds all our original requirements, using nothing more than standard Java or QuickTime for Java classes and methods.

Let's consider briefly how to extend JaVeez. In particular, let's see how to install a movie controller action filter procedure to receive and respond to movie controller actions. For the moment, we'll handle only the mcActionControllerSizeChanged action, which lets us know that the controller has changed size. We'll use that action to support resizing of movie windows.

Resizing Movie Windows

You may recall from our first article on Java and QuickTime that we initially configured our movie windows so that they cannot be resized by the user; we did that by adding this line of code to the JaVeez class constructor:

```
setResizable(false);
```

Now you might think that part of what we need to do in order to allow resizable windows is to change that false to true. Unfortunately, doing that would lead to problems, because AWT would want to draw its own grow box in the lower-right corner of a movie window, as shown in **Figure 14.**

Figure 14: *A movie window with the AWT grow box*

Instead, we want to use the grow box provided by the QuickTime movie controller, which you can see in **Figure 15**.

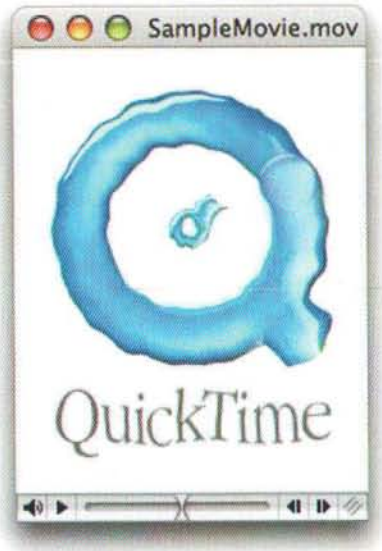


Figure 15: *A movie window with the movie controller grow box*

To accomplish this, we'll continue to pass false in the call to setResizable. We'll get the movie controller to draw its grow box in the standard way, by defining a non-empty grow box rectangle. In the createNewMovieFromFile method, we'll execute these lines of code:

```
QDRect rect = new QDRect(32000, 32000);
mc.setGrowBoxBounds(rect);
```

Once we've done this, the movie controller will draw the grow box in the appropriate location and allow the user to resize a movie by click-dragging in that box.

Handling Movie Controller Actions

Now we need to resize the Java frame (that is, window) when the user resizes the movie using the grow box. As usual, we do this by suitably responding to the mcActionControllerSizeChanged action. First, we need to define a movie controller action filter method. **Listing 9** shows the code we'll add to JaVeez.

Listing 9: Defining a movie controller action filter method

execute

```
public class MCActionFilter extends ActionFilter {
  public boolean execute (MovieController mc, int action) {
    if (action ==
            StdQTConstants.mcActionControllerSizeChanged) {
      pack();
    }

    return false;
  }
}
```

As you can see, we simply call the pack method when we receive the appropriate action. We also need to implement the getPreferredSize method, which is called by pack. **Listing 10** shows our getPreferredSize method.

Listing 10: Setting the size of a movie window

getPreferredSize

```
public Dimension getPreferredSize () {
  Dimension frameDim = new Dimension(0, 0);

  try {
    QDRect rect = m.getBox();

    if (mc.getVisible())
      rect = mc.getBounds();


    // make sure that the movie has a non-zero width;
    // a zero height is okay (for example, with a music movie with no controller
bar)
    if (rect.getWidth() == 0)
      rect.setWidth(this.getSize().width);

    // resize the frame to the calculated size, plus window borders
    frameDim.setSize(
          rect.getWidth() + (getInsets().left +
                             getInsets().right),
          rect.getHeight() + (getInsets().top +
                             getInsets().bottom));
  } catch (QTException err) {
    err.printStackTrace();
  }

  return(frameDim);
}
```

There's nothing much new here. In fact, this method is based on the sizeWindowToMovie method that appeared in the first article in this series. Using getPreferredSize is by far better Java code.

Finally, we need to install the action filter method, like this:

```
mc.setActionFilter(new MCActionFilter(), false);
```

With these changes, the movie windows displayed by JaVeez should resize as expected.

Conclusion

QuickTime for Java provides a very nice set of capabilities for building QuickTime applications with Java. The support for displaying and controlling movies using movie controller components is extensive and easy to use. And, as we've seen in detail in this article, it's extremely easy to write our applications in such a way that they are easily portable to Windows computers running QuickTime and Java. Whether we use Java's reflection APIs or resort to the stub classes, we can get our OS X application running on Windows in a matter of minutes. Cross-platform portability is of course one of the key promises of Java, and it's refreshing to see that promise realized so fully in this case.

Acknowledgements

Thanks are due once again to Anant Sonone for his assistance and support. The OSXAdapter class is based upon sample code by Matt Drance, who also graciously provided advice on these issues.

REVIEW

By William Porter, Polytrope, LLC

Mailsmith 2.1

E-mail client for control freaks

Introduction

If I were writing this review for MacHome, or even MacWorld, I'd spend time talking about how smart and user-friendly Mailsmith is. I would go out of my way to claim that it isn't just for power users, programmers, hackers and geeks. This would be true. It's not.

But I am writing this article for MacTech. You are a MacTech reader. I am going to make the not unreasonable assumption that you *are* a power user, programmer, hacker and/or geek, that you are, in short, a control freak. Bare Bones, developer and publisher of Mailsmith, describes it as an e-mail client written by Mac users for Mac users. I want to describe it more aggressively, as an e-mail client written by control freaks for control freaks. Like you.

I am also going to assume that you are not already using Mailsmith, and that you may need more than just a rundown of what's new in 2.x. Now there is a lot that's new. In fact, in versions 2.0 (released fall 2003) and 2.1 (released January 2004), in addition to hundreds of problem fixes, Bare Bones added hundreds of new features or feature enhancements, including PGP support, considerably enhanced integration with OS X, native support for SpamSieve, greatly improved notification options (even support for the Griffin PowerMate), changes to the default mailbox structure, and much more. If you want all the details, you can get them from Bare Bones' web site. But in this review, I am not going to focus on what's new. I'm going to focus on how you, the power user, can make Mailsmith do your bidding.

Customize!

Let me start by noting that there's a lot you can do to customize the way Mailsmith looks, how it works, and how you interact with it. There are different ways to view mail lists and messages. You can change fonts, font colors, and font sizes for lists, for reading messages, for composing messages, for printing. Window size, text wrap on/off, where your favorite AppleScript editor is located, a slew of options relating to spam, attribution line for text quoted in replies—you can control all these, and more. What I particularly like is the ability to add or edit your own keyboard shortcuts, not just for every command in the menus, but also for palette items such as glossaries, stationery, and scripts. I touch my mouse very little when I'm working in Mailsmith.

Getting and Sending

One option Mailsmith does not give you: IMAP. Bare Bones says that IMAP support is on the list for a future version. But, at the moment, Mailsmith is aimed at the ninety-plus percent of us that don't need IMAP.

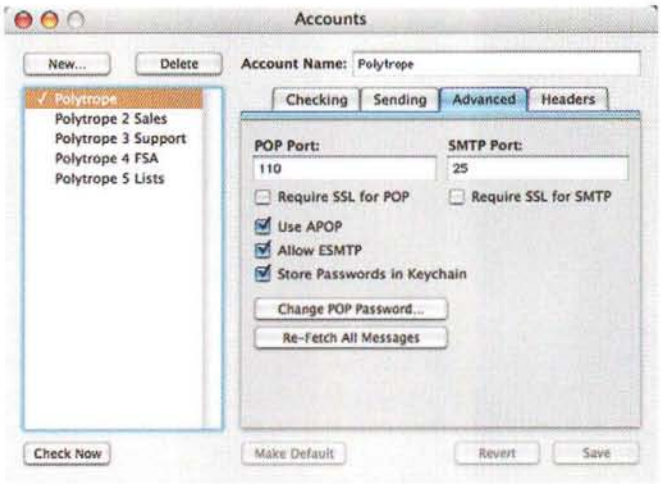


Figure 1. *The advanced tab in the accounts dialog.*

That said, Mailsmith 2.1 gives you a lot of control over your POP and SMTP accounts. You can, of course, have as many different user accounts as you like, each independently configured. Usernames and passwords can be stored in the OS X Keychain. Version 2.x adds a number of valuable new features, including secure (SSL) connections, SMTP authentication, the ability to delete messages from the server after *n* days, faster uploading and downloading, improved

William Porter, Ph.D., a former professor of Classics and Honors at the University of Houston, now spends most of his time doing e-mail, or building databases. He is the owner of Polytrope, LLC, an independent database development firm in Dallas, Texas. He has written for MacTech in the past about 4D, and FileMaker Pro. He reviewed Mailsmith 1.5 in 2002 for TidBits, and later wrote the chapter on filtering in the Mailsmith 2.0 user manual. You can reach him at wp@polytrope.com.

handling of signatures, and more. (See **Figure 1**.) If you have PGP 8 or higher installed, Mailsmith 2 added the ability to cryptographically sign outgoing messages. The POP Monitor, improved in version 2, lets you view messages directly on the server and get them or delete them.

By default now, Mailsmith 2 uses the OS X Address Book to store e-mail addresses. Integration with the Address Book is so smooth, it's nearly invisible.

HTML and "Enriched Text"

Mailsmith does not display "enriched text," that is, font and simple paragraph formatting that may have been added to a message that was written in a program such as Apple's OS X Mail. If someone sends you an enriched-text message, Mailsmith will display it as plain text.

On the other hand, Mailsmith does support full-blown, properly formed HTML messages. To be sure, it does not encourage HTML e-mail. If your mail consists largely of spam, or if the majority of your correspondence is with people who can't express themselves adequately in a single font, well, Mailsmith might not be the best choice for you. But if all you need to do is view the occasional stray piece of non-spam HTML mail, Mailsmith can do it fine. First, it does a great job of extracting the non-HTML content from messages and displaying this in the message window. This lets you see what the message is about without actually having to *see* what the message is about, if you get my drift. And if you decide you need to see the message in all its decorated splendor, you simply click a button, and Mailsmith passes the message to your default web browser. Mailsmith 2.1 integrates beautifully with Safari in OS X v10.3 (Panther).

I can't help noticing that BBEdit 7.1 now offers HTML preview right inside the app. Is something like this in Mailsmith's future? I do not have any inside knowledge, but I doubt it. It's not hard to switch to a real browser, and you would need to do this anyway if the HTML message being rendered contains clickable links to other web pages. And in any case, Mailsmith was obviously designed by and for people who *believe* that e-mail should be plain text.

HTML aside, Mailsmith displays photos and even movies fine, not inline, but right in the program, via integration with QuickTime.

Text Editing Heaven

I remember that, when I first saw Mailsmith version 1.0, I thought it was little more than BBEdit with POP and SMTP support tacked on. Even then, that was both unfair, and unobservant. But ironically, while I have come to appreciate how deep Mailsmith's feature set truly is, I've also decided that Mailsmith's text-handling tools are among its very strongest features. This is an e-mail client for users who write a lot.

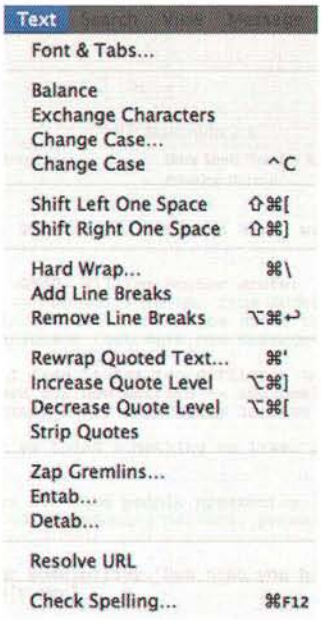


Figure 2. *Mailsmith's Text menu, showing the array of text-editing options. Look familiar? It's identical to BBEdit's Text menu.*

When I have to use another mail client, whether it's Apple's OS X Mail, or Eudora, the web mail app on .Mac or something else, the thing I miss the most about Mailsmith is the rich set of compositional options it provides. Heck, I miss these tools even when I'm working in my favorite word processor, Mellel. In addition to the options shown in **Figure 2**, Mailsmith provides multiple clipboards, unlimited undos (and redos), commands that let you insert a list of a folder's contents, or a file path. Mailsmith makes it easy to define, and use stationery in your correspondence, ditto for signatures.

Best of all, Mailsmith gives you the same powerful glossary feature that BBEdit has. For starters, the glossary feature lets you store, retrieve, and insert bits of reusable text. I use glossaries to store a lot of the things I need to say over and over again in my correspondence with clients: instructions on updating solutions, downloading files from our server, etc.. But, Mailsmith glossaries can go beyond inserting static text through the use of "placeholders." Some placeholders are simply variables that will be replaced by the current date, time, or username. Some placeholders let you manipulate text. For example, I use this glossary with the #SELECT# placeholder to put angled brackets around a selected URL:

```
<#SELECT#>
```

I select the URL, then type option-command-U. (That's the keyboard shortcut I've defined for this glossary). Finally, some placeholders can actually trigger calculations. The "#SCRIPT *scriptname#*" placeholder that runs the script named as a parameter when the glossary is processed. And the "#SYSTEM *command#*" placeholder lets you send Unix commands to your

default shell when the glossary is processed! The SCRIPT and SYSTEM placeholders must return text values, or values that can be coerced to text; the value returned is inserted into the message where the placeholder was originally.

I'm going to pass quickly over one of Mailsmith's most powerful text-processing features, namely, its find-and-replace feature, which, like the same feature in BBEdit, gives you all the power of regular expressions (grep). I use grep in BBEdit fairly often to massage large sets of raw data. I have found that I don't use it very often in Mailsmith.

By the way, Mailsmith provides tools for manipulating text in incoming messages as well. Ever get a message that's been forwarded two or three times and has gotten horribly misformatted? You can't save edits to incoming mail, but you can clean up formatting disasters in two seconds to make messages easier to read. Get mail lists in digest format? Mailsmith 2 will now extract and display the individual messages for you, and you don't need to use a plug-in. And while some might criticize Mailsmith's user interface as austere, those of us who love the program see the same thing, but regard it as a virtue. I prefer to describe it as uncluttered, and elegant. It's a great environment for reading. There's something grown-up about it. By contrast, when I crank up Mail, I always feel like I'm in the Dr. Seuss section of the library.

Message Management

Those who write a lot of e-mail tend to get a lot in return. Mailsmith helps you manage your mail store with power, intelligence, and efficiency. There are scores of nice features to the user interface that make a difference here. Mailsmith gives you a couple different ways to view your mail (either using a unified three-pane browser, or using a separate window for every mailbox). Contextual menus provide access to most of the commands that can be applied to a message selected in a list. Messages can be moved "manually" from one mailbox to another entirely from the keyboard. But, the three keys to controlling your messages in Mailsmith are filters, advanced queries, and its built-in support for Michael Tsai's SpamSieve.

Since I've written about Mailsmith's filters at length elsewhere, I'll restrict myself here to the executive summary: Mailsmith's filtering options are without equal. Mailsmith filters can have as many criteria as you like, and the criteria can be as simple, or as sophisticated, as you like—almost anything is possible, from finding messages from your mother, to grep pattern matching. Filters can also have multiple actions, such as applying a label to a message, moving the message to a folder, sending an automatic reply, or running an AppleScript script. And, since filters can be attached not just to the incoming mailbox, but to *any* mailbox, they can be triggered automatically whenever a message is moved, something not possible in other programs. My sense is that the designers of Apple's OS X Mail believe that users are not going to bother with filters; that's why filters have been stuck away in the preferences dialog. In Mailsmith, however, filters are front and center. In fact, there are three quite distinct ways to define filters, all ready to hand.

Mailsmith's simple query command is great for finding messages in the current mailbox from senders whose name include "Bob". But the advanced query command can find almost anything. Actually, the advanced query dialog is nearly identical to the criterion part of the filter-definition dialog, but you may find yourself using different aspects of the dialog in queries than you use in filters. I don't have any filters that test for dates, but I query on dates fairly often. The advanced query feature has two significant limitations. You cannot tell Mailsmith to search within multiple, non-nested mailboxes (something you can do in Eudora). And, advanced queries cannot be saved.

I used to filter incoming messages very aggressively as soon as they arrived, and Mailsmith 2 will filter messages into appropriate mailboxes, then open mailboxes that have new mail. But nowadays I let incoming messages sit in the incoming mailbox, so I can read them all in one place. If I need to find a thread, I use the advanced query dialog (or a script) to do so. When I'm done reading, I apply filters manually. I let filters automatically delete list messages that I haven't marked for keeping. I used to have lots of individualized mailboxes with limited numbers of messages. Now I find it easier to deal with fewer, fatter mailboxes. The change in my M.O. reflects the fact that the advanced query command has become a routine part of my interaction with the application.

Finally, of course, there's the headache of spam. Apple Mail has a powerful spam filtering feature built-in. Most other OS X e-mail clients can use Michael Tsai's SpamSieve scripts to achieve a very similar degree of success. The great thing about SpamSieve is that you train it yourself, and you control every aspect of its operation. And SpamSieve, like Mailsmith, is brilliantly integrated with OS X. You can, for example, use your Address Book as a white list. The smart developers at Bare Bones, recognizing that SpamSieve was a winner, decided that the best thing they could do was provide native support for it in Mailsmith, so that is what they did. This gives Mailsmith the advantage over the non-Apple competition, because SpamSieve doesn't need scripts to work in Mailsmith, and also because it can find, and remove spam before any filtering is done on your messages. SpamSieve has the ugliest icon in my Dock, but doggone it, it is effective. It catches close to 100% of my spam, with just one false positive in the last six months.

Finally, AppleScript

Like BBEdit, Mailsmith provides deep, and extensive support for AppleScript. You want control? AppleScript gives it to you. You can, in essence, create your own meta-app with AppleScript to make Mailsmith jump through hoops.

Mailsmith is both recordable, *and* attachable. Recordability works great sometimes. I wanted to write a little script to select text from the insertion point to the end of the sentence. I turned "Record" on in the Scripts menu, opened the Find definition dialog, enabled grep, and typed in a simple regex that finds the next period or other common sentence terminator. Executed the find, then stopped recording. Here's what was generated:

```
tell application "Mailsmith"
  activate
  find "[\\.\\!\\;\\?\\r]" searching in text 1 of message
```

```
        window 1 options {search mode:grep, starting at
        top:false, wrap around:false, backwards:false, case
        sensitive:false, match words:false, extend
        selection:true} with selecting match
end tell
```

Worked perfectly. Note the use of *two* backslash characters. The second "\" escapes the reserved character that follows it ("." and the others), and is required by Mailsmith. The first "\"escapes the second, and is required by AppleScript.

Now, recording does not always work this well, but many of the kinds of things I personally want to do with AppleScript are child's play. Here, for example, is my "quit" script, triggered when I type Command-Q. (Yes, I've reassigned the keyboard shortcut to this script.)

```
tell application "SpamSieve"
  activate
  quit
end tell

tell application "Mailsmith"
  activate
  quit
end tell
```

By the time you read this review, Bare Bones may have fixed the bug that makes it currently impossible to attach a script to the Quit command in the Mailsmith menu. When that bug is fixed, I'll get rid of the second part of the script above, and attach this script to the quit command directly. Then it will be run regardless of whether I type Command-Q or invoke the quit command from the menu.

I mentioned above that you can't save advanced queries; you can, however, script them. AppleScript is a great way to fill in the rare gaps in Mailsmith's feature set. Here, for example, is a script I picked up from another user, and adapted just a bit. The script finds messages that have the same subject as a selected message.

```
tell application "Mailsmith"
  try
    set message_list to selection as list
    set the_message to item 1 of message_list
    set the_container to container of the_message
    set the_subject to subject of the_message
    if the_subject starts with "Re: " then
      set the_count to count characters of the_subject
      set the_subject to characters 5 thru the_count ¬
        of the_subject as text
    end if
    set the_name to "Thread: " & the_subject as text
    make new mail list window with data every message ¬
      of the_container whose subject contains the_subject ¬
      with properties {name:the_name}
  end try
end tell
```

This script compensates for Mailsmith's lack of a true threading feature. By comparison, Apple's OS X Mail *does* have a nifty threading feature, but it hardly compensates for the many serious limitations in Mail's support for AppleScript.

If you are already an AppleScript adept, you are going to love what you can do with Mailsmith. And if you're not? Well, neither am I. But as I suggested above, there's a great deal you can do with very simple scripts, and many more complicated scripts can be borrowed. And, Mailsmith may inspire you to learn more. Matt Neuberg's new book on AppleScript is my nightly reading these days. My hope is someday soon to write an AppleScript that will not only get my mail, and read it for me, but draft and send thoughtful replies as well. I really need to get away from the computer a bit, maybe walk the dog.

System Requirements and Purchasing Information

Mailsmith 2.1 requires Mac OS X version 10.1.5 or higher; OS X version 10.2.6 or later is strongly recommended. (Integration with the OS X Address Book requires 10.2 or later.) You can read more about Mailsmith's features and download a fully-functioning thirty-day demo from Bare Bones's web site:

http://www.barebones.com/products/mailsmith/index.shtml

Mailsmith 2.1 costs $79 if purchased as a new product directly from Bare Bones. Upgrade and cross-grade pricing is also available.

SpamSieve is shipped with Mailsmith but separately licensed. You can more information here:

http://www.c-command.com/spamsieve/

The latest version of SpamSieve as of this writing was 2.1.1.

References

In 2002, I wrote three articles on Mailsmith for TidBits, a review of Mailsmith 1.5 (http://www.tidbits.com/tb-issues/TidBITS-638.html), and a pair of articles on distributed filtering (http://db.tidbits.com/getbits.acgi?tbser=1227). I say a lot in those earlier articles that is still pertinent to Mailsmith, and which I didn't want to repeat here. I recommend the earlier review in particular. The articles on distributed filtering later metamorphosed into chapter 9 of the Mailsmith 2.0 user manual.

Bare Bones has a web page intended to list AppleScript scripts for use with Mailsmith, here:

http://www.barebones.com/support/mailsmith/script_library.shtml

As of this writing, only two scripts are listed. Many more scripts can be found here:

http://www.mailsmith.org/scripts/

The Mailsmith Talk list, sponsored by Bare Bones, is frequented by AppleScript mavens, and they're generous in sharing their knowledge. The threading script used as an example above was adapted from scripts shared on the Mailsmith Talk list.

Matt Neuberg's new book is *AppleScript: The Definitive Guide* (O'Reilly, 2004). Matt hit a homerun with his earlier book on RealBasic; he's hit another home run with this one. As it happens, Matt reviewed Mailsmith 2.0 for TidBits (http://db.tidbits.com/getbits.acgi?tbart=07289). I recommend his review, which takes a quite different tack from this one. It is entitled, "True Confessions of a Mailsmith Switcher."

By Kevin Hemenway, Imitating Conspirator

Programming and MySQL

Modifying our MySQL database with the shell and PHP.

We finally added information into our MySQL database last issue, but in a rather infantile way: by writing all the SQL statements (like those seen in **Listing 1**) into a text file and then passing them to MySQL with a command line, mysql mactech < mactech-insert.sql, which completes silently when successful. This is certainly helpful if we're passing default information to be initialized in newly created databases, but not very useful if we want to programmatically access the information within.

Listing 1: Three new SQL INSERT statements for our database.

mactech-insert.sql

```
INSERT INTO books
   SET publication = "2000-00-00",
       title = "Object Oriented Perl";

INSERT INTO books
   SET publication = "1999-00-00",
       title = "MyEssQueEll";

INSERT INTO books
   SET publication = "2003-00-00",
       title = "PHP and MySQL Web Development";
```

In this article, we'll talk about two more ways to manipulate data: through the MySQL interactive shell, and via PHP's built in database functions. First up, let's walkthrough the interactive shell, which is quite helpful for quickly testing out new SQL statements or applying immediate fixes without new code overhead and development.

The MySQL Interactive Shell

Starting up the MySQL interactive shell is mindlessly simple. Just type mysql and you'll be shown a mysql> prompt, similar to the shell prompt you see when you first enter the Terminal. Since we've yet to specify a database, we're in a sort of limbo: MySQL knows we're here, but we've yet to tell it anything of import. First we'll show which databases have been configured, choose which one we'd like to work with, and then ask for a table listing:

```
:~ > mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.15

Type 'help;' or '\h' for help.
Type '\c' to clear the buffer.

mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mactech  |
| mysql    |
| test     |
+----------+
3 rows in set (0.26 sec)

mysql> USE mactech;
Reading table information for completion of
table and column names. You can turn off this
feature to get a quicker startup with -A.

Database changed
mysql> SHOW TABLES;
+-----------------+
| Tables_in_mactech |
+-----------------+
| books           |
| person          |
| relationships   |
+-----------------+
3 rows in set (0.05 sec)
```

The most obvious fact from the above is that all SQL commands must end with a semi-colon. If you happen to forget that termination, MySQL will change its shell prompt to show you it's waiting patiently for a complete command. The following is the same command as before, only split with new lines. Notice the prompt indicating an incomplete statement:

```
mysql> SHOW
    -> TABLES
    -> ;
```

Kevin Hemenway, coauthor of *Mac OS X Hacks* and *Spidering Hacks*, is better known as Morbus Iff, the creator of disobey.com, which bills itself as "content for the discontented." Publisher and developer of more home cooking than you could ever imagine (like the popular open-sourced aggregator AmphetaDesk, the best-kept gaming secret Gamegrene.com, the ever ignorable Nonsense Network, etc.), he'd rather be nursing his wounds with a swift kick to the head. Contact him at morbus@disobey.com.

A good portion of readers will know that the shell opened by the Terminal gives you command line history (press the UP arrow to see commands you've previously typed) as well as file or directory completion (press TAB after typing the first few letters). MySQL also supports these time-savers: pressing UP will show you SQLs you've typed previously, and pressing TAB when typing a table or column name will auto-complete the nearest match.

In our previous work with SQL, we've touched on the INSERT, UPDATE, and DELETE commands. We've yet to touch on the most often used, however, which is SELECT. While our previous statements have focused on adding, modifying, or deleting data, the sole purpose of SELECT is for displaying:

```
mysql> SELECT * FROM books;
+----+-------------------------+-------------+
| id | title                   | publication |
+----+-------------------------+-------------+
|  1 | Spidering Hacks         | 2003-11-01  |
|  2 | Mac OS X Hacks          | 2003-04-01  |
|  3 | Object Oriented Perl    | 2000-00-00  |
|  4 | MyEssQueEll             | 1999-00-00  |
|  5 | PHP and MySQL Web Development | 2003-00-00 |
+----+-------------------------+-------------+
5 rows in set (0.00 sec)

mysql> SELECT title, name FROM person;
+-------+---------------+
| title | name          |
+-------+---------------+
| Mr.   | Kevin Hemenway |
+-------+---------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM relationships WHERE book_id = '2';
+-----------+---------+
| person_id | book_id |
+-----------+---------+
|         1 |       2 |
+-----------+---------+
1 row in set (0.05 sec)
```

The above shows three different variants of a SELECT statement—many more are possible. The first is the easiest to understand: "select everything from the books table". The second is an example of specifying only the columns you want to see, in any order. Even though title came after name in our original CREATE statement (last issue), SELECT allows us to reorder things however we decide best. The third statement is an example of more intimately specifying which exact rows you'd like to retrieve.

Comparing the above walkthrough to the command line we entered last issue (mysql mactech < mactech-insert.sql) gives us a better understanding of what's going on. We specify the database to connect to (mactech; similar to USE mactech in the MySQL shell), and then send a bunch of SQL commands in a batch, as opposed to manually entering them one at a time. It doesn't take a giant leap of faith to realize that we can save the USE database step by specifying it on the command line:

```
~ > mysql mactech
Reading table information for completion of
table and column names. You can turn off this
feature to get a quicker startup with -A.

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 4.0.15

Type 'help;' or '\h' for help.
Type '\c' to clear the buffer.

mysql> UPdAtE books SET title = "MySQL"
    -> where title = "MyEssQueEll";
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SeLeCT id, title FRoM
    -> books WHERE title LIKE '%SQL%';
+----+-------------------------+
| id | title                   |
+----+-------------------------+
|  4 | MySQL                   |
|  5 | PHP and MySQL Web Development |
+----+-------------------------+
2 rows in set (0.30 sec)
```

The previous listing shows an example of entering the MySQL interactive shell with a database already selected, updating a row of misspelling in our previous INSERT (**Listing 1**), and then getting a pattern match with LIKE. The % characters are boundary placeholders for "anything or nothing", so our final statement finds the letters "SQL" in the beginning, middle, or end of a title. Changing "SQL" to "my" would return the same set of results, confirming that LIKE searches are case-insensitive.

Up until now, all our SQL commands have used capital letters, but you'll notice that they too are case-insensitive. For clarity, I prefer uppercase SQL: it just makes things easier to mentally and visually parse after long hours of fevered coding.

Accessing MySQL From PHP

Our next step is to access our mactech database programmatically with PHP. You'll notice some similarities with the above interactive shell process: we connect to MySQL, choose a database, and issue some SQL queries. You'll note that we're passing the MySQL username and password we created with mysql_setpermission a few columns back. This is important for security: just like you don't want your sister messing with your personal files, you don't want to make it too easy for web interlopers to affect the various databases you maintain. In most cases, once you're finished developing an application, you'd tighten the user's permission even further (say, to restrict DELETE and DROP access).

Save the contents of **Listing 2** into /Library/WebServer/Documents:

Listing 2: PHP code for accessing our MySQL database.

db_access.php

```
<h1>MySQL Database Access in PHP</h1>
```

```
<?php
    $dbh = mysql_connect("localhost","favemarksman","****")
      or die ("There was an error connecting to MySQL.");
?>
```

Before we delve deeper into our code, I wanted to show you a one-character-difference that can be used to improve security. Astute readers will notice that even though I meant davemarksman (the MySQL user created a few issues ago) I mistyped as favemarksman:

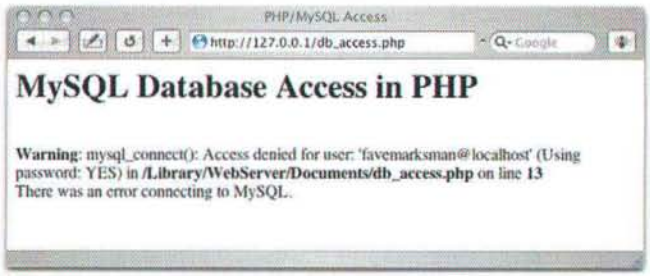


***Figure 1**: An error occurred during the database connection.*

Here's the problem: if someone comes to our site and triggers this error message, we've freely given them four pieces of valuable information that can be used against us for exploitive purposes. We've told them we're using MySQL as a database, that it's installed on the same machine as the web server, that there's (possibly) a user named favemarksman (a similar error would occur if the username was correct, but not the password), and revealed a directory path (which, in this case, infers we're using OS X, a fifth fact).

Listing 3 contains a much stronger version of our code, which removes any mention of the technology being used, and stops MySQL from spitting useful information to our visitors. The real magic happens with the @ symbol before our function name: using it will silence any automatically visible errors. This makes for more secure code, as well as a more professional web site (i.e., what impression does it leave visitors if they see nothing but errors?)

Listing 3: Revised PHP code for accessing our MySQL database.

db_access.php, revised

```
<h1>Database Access in PHP</h1>

<?php
    $dbh = @mysql_connect("localhost","favemarksman","****")
      or die ("An error has occurred. Please report this.");
?>
```

The revised output is shown in **Figure 2**:

***Figure 2:** No more information disclosure. Much better.*

With that out of the way, we can now accomplish something. **Listing 4** contains complete (but simple) PHP code to insert some information into the mactech database, view it, and then DELETE a bit. As with most programming, there are many ways this can be done: more information about the different mysql_ functions we've not shown can be found at the PHP web site: http://www.php.net/manual/en/ref.mysql.php.

Listing 4: Finished PHP code for accessing our MySQL database.

db_access.php, finished

```
<h1>Database Access in PHP</h1>

<?php

  // connect to the database server.
    $dbh = @mysql_connect("localhost","davemarksman","****")
      or die ("ERROR: Could not connect to the database!");

  // choose our database.
    @mysql_select_db( "mactech" )
       or die ("ERROR: Could not select our database!");

  // create a SQL statement. Notice that through
  // PHP, the SQL terminating semi-colon is optional.
    $statement = "INSERT INTO person SET
                    name          = 'Dave Mark',
                    date_of_birth = '1901-03-31',
                    title         = 'Intern',
                    designation   = 'Bullseye Hole Filler'";

  // standard way of executing SQL through PHP.
    $response = @mysql_query( $statement, $dbh );

  // mysql_error() would give too much information for
  // a production site, but this is just an example.
    if (!$response) { print mysql_error () . "\n"; }

  // create and execute another SQL statement.
    $sg_made_out_of = "SELECT * FROM person;";
    $response = @mysql_query( $sg_made_out_of, $dbh );

  // this is one of a few ways to iterate through rows.
    print "<h3>People, List #1</h3>";
    while ( $person = @mysql_fetch_array( $response ) ) {

      // column name is array key.
        print "$person[id], $person[title],
               $person[name], $person[designation]<br />";
    }

  // one more SQL statement, this time a delete.
    $freedom = "DELETE FROM person WHERE name LIKE '%Mar%'";
    $response = @mysql_query( $freedom, $dbh );
```

```
// and make sure he's really gone.
  $sg_made_out_of = "SELECT * FROM person;";
  $response = @mysql_query( $sg_made_out_of, $dbh );

// another way of iterating through rows, only
// as variables, not hash keys. Could get messy.
  print "<h3>People, List #2</h3>";
  while ( $person = @mysql_fetch_array( $response ) ) {
      extract ( $person ); // make columns variables.
      print "$id, $title $name, $designation<br />";
  }

?>
```

The results of running this script are in **Figure 3**. Since we're inserting a new record, then deleting it, successive loads will assign the new record an ever-increasing sequential ID, even though the data we're entering is exactly the same.

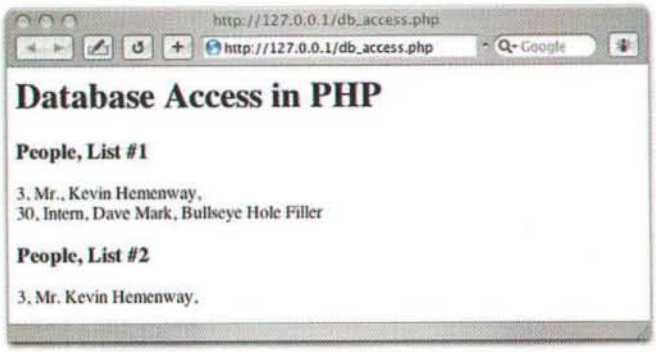


Figure 3: *The final results of our database access in PHP.*

Homework Malignments

As is typical in a four-page article, we've barely touched the surface of what PHP can do in regards to database access and handling. Next month, we'll change gears and see how Perl handles the same logic. Until then, contact the teacher at morbus@disobey.com.

- For more information about secure coding with PHP, check out the following articles: "A Study In Scarlet: Exploiting Common Vulnerabilities in PHP Applications" from Shaun Clowes (http://www.securereality.com.au/studyinscarlet.txt), and "Ten Security Checks for PHP" from Clancy Malcolm (Part 1: http://www.onlamp.com/pub/a/php/2003/03/20/php_security.html and Part 2: http://www.onlamp.com/pub/a/php/2003/04/03/php_security.html).

- Since you'll be at O'Reilly's OnLAMP site anyways, check out the rest of the PHP DevCenter at http://www.onlamp.com/php/. The "PHP Foundations" (http://www.oreillynet.com/pub/au/135) series has a three-part article on "PHP Security", as well as recent crash courses for MySQL (which, in hindsight, uses the same clichéd "person/book" example I did. Excelsior!)

- For more on the @ error silencer, see http://us3.php.net/operators.errorcontrol.

By Chris Kilbourn

Business Planning

My first business plan was laughable.

There were ridiculous assumptions, glaring omissions and there were more marketing fluff words in it than actual substance. Even though I cringe when I read it today, it still put me significantly ahead of the game when talking to bankers, potential investors, and clients. Why? Because I actually had a plan to show them, flaws and all.

Like a college degree, a business plan neither guarantees future success, nor proves that you have mastered a given subject, but it does show that you put in the time to think about your business, and how it will operate. The time spent writing your plan will force you to think about, and explore all the facets of your business. It will help you to uncover hidden assumptions, crystallize how you will actually make money, and assist you in developing realistic goals for your business.

I am still amazed at the number of entrepreneurs I meet who feel that writing a business plan is not a good use of their time. Their excuses range the gamut from claiming to not have enough time to write it to blithely claiming that they do not need one because the business is doing well without one.

If you ever expect to open a line of credit with a bank or vendor, raise money from private individuals or venture capitalists, or recruit senior management for your company, you will need a business plan. Without a business plan, you simply will not be taken seriously by those people, and you will be wasting everyone's time, including your own.

What Should Be In Your Business Plan

A business plan should answer what I like to call the Golden Quad:

- What will the business do?
- Where will your customers come from, and how will you sell to them?
- How will the company make money?
- Who will run the business, and why are they qualified to do so?

I call them the Golden Quad because without answering them in your plan, you will likely never strike gold in your business.

As you work through writing your plan, continually ask yourself if you are answering these four questions clearly, and directly. For every sentence that is in the plan, if it does not speak to or support the Golden Quad, do not include it.

I have read plans with insightful analysis of markets, and wonderful descriptions of cutting-edge technology, and that were very well-written, but didn't tell me a damn thing about the company's product or service, who their customers were, how they were going to make money, and why the managers were qualified to lead the company. In short, they were a great read, but useless as plans.

I would be remiss unless I emphasized the importance of the financial component of the business plan. Your plan should include projected cash flow and income (profit and loss,) statements, a balance sheet, a listing of all capital (expensive) equipment, and a list of the assumptions that went into the projected financial data.

To a person, the bankers and investors I have met looked at my financial data before reading the plan, or executive summary. If your financial numbers do not add up, are unclear, or your assumptions are incorrect, your plan will not be read. Period.

The good news is that there are many free tools for business financial planning available. The Small Business Administration provides a wealth of information in this area on their web site, for example. If numbers are not your strong suit, you will want to hire or contract for some help in this area.

Who Are You Writing The Plan For?

Your business plan is being written for a variety of audiences. First and foremost, you write it for yourself. You may have spent months or even years thinking about starting your business, but I'd be willing to bet you spent most of

Chris Kilbourn is an independent small business, network and web infrastructure consultant. Chris is also the founder of digital.forest, Inc., http://www.forest.net, which offers database, application and web hosting services in addition to server colocation. When he's not out running marathons, you may contact him at chrisk@forest.net.

that time thinking about what you were going to do, not how much floor space, and equipment each employee would require.

Your business plan will become your guide, your to-do list, and your touchstone when you lose your way. With the myriad details that must be attended to in any business, re-reading my business plan on a periodic basis became a ritual for me. It was the compass that let me know if I was still heading in the right direction.

Secondly, you write your business plan for your employees. Post dot-com crash, savvy interviewees asked to see our business plan, which we gladly furnished to them, sans financial information. A clearly written, and shared business plan shows potential, and existing employees that management has thought through the business, and is willing to be held accountable to it.

By sharing your plan with your employees, you invest them in the vision of where the business is going. If you hide it from them, which seems to be an unfortunate practice at far too many businesses, management directives tend to be questioned more vigorously. Think about it: if you were following someone into unknown territory, would you be more or less inclined to question their judgment and authority if they shared with you up front where they were planning to go? As an added bonus, you gain the benefit of employee input into the plan.

You write your business plan for your bankers and vendors. Bankers and vendors who extend credit must perform a risk assessment before deciding to grant you credit. Providing them your business plan with a sound financial model in it helps them to understand the financial risk they will be taking with you. From personal experience I can tell you that no plan, or one that has unclear financial information in it, will prevent you from receiving credit which can significantly inhibit the growth of your business.

Finally, you write your plan for your investors. Unless you are a seasoned entrepreneur with several business successes under your belt, the only people you will be able to raise money from without a plan will be your friends and family. The business plan also helps you create your 'elevator pitch.' This is a 3- to 5-minute verbal pitch that answers the Golden Quad, and hopefully hooks a potential investor to learn more about your company.

We will cover fundraising in a future column, but a truism in fundraising is that people invest in people not plans, but without a plan, an investor will not invest with a person. Period. No plan = no investment money.

How Long Should The Plan Be?

You should create three versions of your business plan, and each version will vary in length, and content depending upon whom the target audience is.

The first version you create should be a detailed full plan that should be as long as it needs to be to answer the Golden Quad, but keep it under 20 pages with supporting materials. Anything longer than that and you risk reader fatigue. If you feel that including a large amount of supporting material is necessary, consider creating a separate document that includes them. Bear in mind that unless you are submitting your plan to a venture capital firm, large amounts of supplemental, and supporting information are routinely ignored.

This detailed plan is for you, for your employees, for investors who express serious interest in investing and for bankers or vendors extending you significant credit.

The second version should be an executive summary no more than 4 pages in length, not counting supporting financial documents. This should be a succinct, boiled-down version of your full plan that provides enough information to clearly answer the Golden Quad.

The executive summary plan is an overview version for potential investors, potential employees and vendors who you plan to work with on a regular basis but for whom the full plan would be overkill.

The third version should be a bullet-pointed slide version of the executive summary. For this third version, select the talking points of the executive summary and then develop a narrative to cover each point.

This slide version is great for sales presentations, and investor pitches. The flexibility of doing it as digital content is that you can tailor your message to each group or individual you present it to.

Sharing Your Plan

As described earlier, you should be sharing your business plan with a variety of people. When it comes to sharing it, exercise some common sense when doing so.

This means that you should have versions that include and omit financial data, proprietary information and competitive analyses. You should also include a version and tracking number on each copy that you distribute and then keep a list of who received which. Each page should be marked, "Copyright [Firm Name.] Proprietary and Confidential - Do Not Distribute," and include the tracking and version number.

Many entrepreneurs are reluctant to share their plans out of a fear that their information could end up in a competitor's hands. In general, this is an overblown fear. Unless you are the target of directed corporate espionage, most people will flip through your plan, and then shove it in a drawer, never to be seen again.

For anyone who is not a potential investor however, you should protect your rights by having them sign a non-disclosure agreement prior to giving them a plan. In the

event that you discover that the information has been shared with parties that should not be privy to your plan, the tracking number of the plan then provides you with strong legal recourse to sue for damages. Consult a lawyer for the finer points of this issue. There have been cases where disclosure of business-plan data resulted in financial settlements that made pursuing the business idea not worth the financial risk!

When it comes to investors you just have to go out on a limb and trust that they will not unduly share your plan. I will cover this process in more detail in a later column, but suffice to say that investors do share plans with others, but it is only in extremely rare cases that this behavior will cause you material harm. In most cases, this is a good thing.

Research and Knowledge – Your Key to a Successful Plan

You will need to do research in order to write your business plan. The depth and quality of your research and how much you learn will be reflected clearly in your plan. The more research that you perform, the easier it will be to write your plan.

I spent close to five months doing research before writing my first business plan. This time was spent reading about as many business failures and successes as I could find, performing market research and financial planning massaging spreadsheets.

Being fully in the camp that forewarned is forearmed, I strongly encourage you study business failures, small and large. Santayana's warning, "Those who cannot remember the past are condemned to repeat it," comes strongly into play here. Businesses tend to fail and succeed for very predictable, well-documented reasons. You should learn what they are before you ever take a dollar from a customer or write your plan. Knowing what the pitfalls and accelerators are will help guide you in crafting your plan.

Questions that you will simply have to answer in your research for your business plan are: How large is your target market in potential number of customers and in dollar value? Where are these customers located and how will you be best be able to sell to them? What equipment will you need? How many employees will you need? Who is your competition? What are the barriers to entry for competitors in your market? Is what you are doing novel and unique, or is it well-known? If it is well-known, how will you make money in an already established market?

There are many other questions that will need to be asked and answered, but they will vary by the industry you are competing in. If the thought of a research project terrifies you, and I've met more than a few people for whom this is the case, there are a wide variety of sources and support that can help you with the process.

The best, and in my opinion, first place to start when you begin your business plan is your local Small Business Administration office or their web site at http://www.sba.gov. The SBA will be able to provide you with business plan samples, checklists, basic market research, networking, and their most valuable resource, SCORE counselors.

The SCORE counselors are retired and working executives, and each of them have been in your shoes before in trying to pull a business plan together. They will sit down with you, and review your plan if you have one, or help guide you in compiling your own. SCORE counselors will provide you with one-on-one, individualized feedback while sharing their knowledge, and experience with you. You should avail yourself of their services.

Another underutilized avenue of research, and knowledge is your local university's business school and associated library. There are legions of MBA students looking for real-world experience in business via internships, and special research projects. In many cases, these services can be had for little to no cost. Students get material for projects, you get data for your plan, and everyone's happy. Business school libraries also contain innumerable business case studies, books on business planning, and stories of business successes and failures.

Not to be overlooked is the Internet. When I was pulling together my business plan in 1994, Internet business resources were few and far between. Today, there is a vast amount of information published by the SBA, business schools, and entrepreneurs. A quick spin through a search engine will present you with a wealth of useful, (and not so useful,) information. The caveat here is to second-source facts and figures found on the Internet with known offline information sources.

Format of the Plan

So how should your plan be formatted? It should have the following sections:

Business Description

This is the 'nut' of what your company does, and how you plan to make money at it. This is your business plan introduction, and it is the bare-bones description of why you are in business, and the general view of the financial opportunity that you are pursuing.

Company Overview

This is a more detailed look at the company. You should describe the company's business structure, provide a general overview of current operations including employee and customer counts, where you are located, the current level of business development, (seed, start-up, established firm, etc.,), and a brief history to date.

Business Opportunity

This part strikes right at the first question of the Golden Quad - what will the business do? Here you should detail how large your potential market is in number of customers, and revenue. The best way to think about structuring this section is to explain what problem you are solving for your customers. Detail the problem, then explain how your company will solve that problem for them, and make money from providing the solution.

Market Analysis

Here is where most of the research you have done will shine through. Demonstrate that you know that your overall market is segmented, and detail which segments you will go after, and why you are going after them as opposed to others. Describe historical trends, and market blips, good and bad, and how you plan to ride them to success.

Strategy

Describe and detail how you will execute your sales and marketing strategy for the customers you identified in the Business Opportunity section, while tying it into the Market Analysis section. This is all about how you will attack your market, take it over, and make money. You will need to explain here the core financial model of the company. Is it direct sales? A software upgrade revenue model? A monthly annuity for services rendered? Lay it all out here.

Competition

Every business has competition, whether they think they do or not. Convince your readers that you understand your competition, that you can penetrate their barriers to entry into their markets, and then erect your own barriers to keep them out of yours. Your strategy should be clearly reflected in how you manage your competition.

Management Biographies

These should be single-paragraph resumes with an emphasis on career achievements that are applicable to the current business endeavor. Your goal here is to convince the reader that your team has the experience, knowledge, and skills to execute the plan successfully.

Summary Financial Data and Projections

Welcome to the land of spreadsheets and numbers. This section should include a 3-year look at actual, and projected cash flow and profit and loss. Income and costs should be broadly grouped for the executive summary version of your plan, and be detailed in the full plan. A balance sheet should also be included if there are already company assets.

Additionally, a list of all the financial assumptions that went into your projections must be included. Good examples of assumptions are rates of inflation, interest, and costs for things like equipment and employees. The more detailed your financial plans, the easier it will be for you, your bankers, and your investors to understand the core financial model of your company; and to adjust it as you move forward.

As an example, digital.forest's financial model is a 2MB, 10-tab spreadsheet where if you adjust the square footage for employee offices, it will automatically update and adjust costs for office space and fixtures and update cash flow and income projections. There are similar 'knobs and sliders' for service costs and revenue that are linked back to the financial reports.

Subscription Agreement

This is a section that should only be distributed with the full business plan, and included when you are fundraising. The subscription agreement lays out the terms and conditions for investors. The subscription agreement should be provided by your legal team. Don't worry about this section right now unless you are actively soliciting large sums from investors.

As a final word on form, print it on a laser printer, not an inkjet, have it bound with a clear plastic cover and have a cover page with your company name on it with your logo,

preferably in color. While the content is what really matters, your plan is more likely to be favorably received if its presentation looks professional.

Reality Check

Surprisingly, one of the most useful things about a business plan that is rarely mentioned is the reality check it provides for your business. Working through the financial, and operational assumptions for the business in your plan will highlight absurdities, and processes that just won't work.

I still clearly recall a digital.forest planning meeting to review our financial model, and ending up with a $3 billion a year revenue stream after four years of operations. Those of us in the room had a good chuckle around the table as we knew that was a completely ridiculous number. It highlighted that there must be a hidden assumption somewhere in the plan that was grossly inflating revenue.

The source identified and fixed, we came back down to Earth with a projected $300 million a year company after four years of operation. (Hey, it was 1999; everyone was thinking that way!) Which only goes to show that even when you think you've figured everything out, and triple-checked your assumptions, you can still be wildly wrong.

It is important to question every result and goal in your plan, even if they seem so rock-solid that it feels like a waste of time. At every major iteration of the digital.forest business plan, we have been able to prune out tacit assumptions that were based on wishful thinking, not on reality. If you don't do it first in the privacy of your office, it is likely that your bankers and investors will do so in a public space in front of people that you are trying to impress.

Lather, Rinse, Repeat

Businesses that have business plans that are not updated on a regular basis tend to be more at risk of failure than businesses without a plan at all. Why? Because a business plan that is not updated can act as a dead hand, guiding the company towards its doom.

The Soviets were fantastic planners. They generated 5-year plan after 5-year plan for things like farm collectivization, and industrial output targets. What they were spectacularly crappy at was updating their plans. Their plans would say they were going to make 75,000 shoes even if all the cows died, and there was no leather with which to make the shoes.

Far too many business people act like Soviets, and feel that once their business plan is written, they're done.

Nothing could be further from the truth. Competitors come and go, technology shifts, inflation goes up or down, things go well, things go poorly. Each of these, and a myriad of others not listed, are excellent reasons to update your business plan.

Without periodically updating your business plan, you run the risk of delivering products, and services your clients do not want or operating your business at a loss because market shifts have changed the profit model for your company.

Andy Grove of Intel says that only the paranoid survive. The reason for that is because the paranoid are constantly re-evaluating the status quo, and adjusting their plans accordingly as they go along.

The Last Laugh

While I still cringe when I re-read my first business plan, over the years it has evolved into a powerful document that has evolved with the times. It has provided direction and purpose to employees, investors judged the merits of it by entrusting their funds to digital.forest because of it, and companies we have acquired felt safe that we would treat their customers, and their staff well because we clearly knew what we were doing - it was written down for all to see.

NEXT MONTH: Accountants, lawyers and bankers - fun people to spend time with; really!

[Do you have a business question that you would like answered? Send it along to chrisk@forest.net

By Ruben Kleiman

The Cool Side of the Moon

The Eclipse IDE on the Macintosh

The Eclipse IDE

In this article we'll tour the Eclipse IDE and discuss how it fits into the Macintosh architecture, focusing on development for Java. We will go beyond hype and precisely look at how Eclipse now stands on the Macintosh.

Why Eclipse?

With so many excellent, even free, IDEs out there, why Eclipse? For the writer, there are three reasons. First, Eclipse is intuitive to learn and use, and just one IDE serves a wide variety of purposes—from Java, AOP, web, to C/C++ development. The investment pays good dividends.

Second, Eclipse's general architecture has created a global community of plug-in developers from whom I draw a wide variety of tools. Some research groups release plug-ins unavailable elsewhere. In a pinch, I can easily roll my own. One way to think of Eclipse is as *Emacs* for the User Interface generation. While Lisp is the language for extending Emacs, for Eclipse the language is the plug-in architecture and Java.

Finally, I'm addicted to Eclipse's extensive and reliable refactoring features: they have saved my overnighters for more useful work than code wrestling.

Features and Benefits

Eclipse is a robust, open-source IDE based on a general plug-in architecture. All components of Eclipse are plug-ins. The most widely used plug-in is the *Java Development Toolkit* (JDT). Another is the new *C/C++ Development Toolkit* (CDT), supporting the GNU gcc compilers, including Objective C. The Eclipse IDE runs on and has cross-platform development libraries for MacOS-Carbon, Win32-win32, Win32-CE/win32, Solaris/motif, QNX/photon, Linux/qt, Linux/motif, Linux/gtk, HPUX/motif, and AIX/motif.

Apart from the Java and C++ editors, there are plug-in editors for UML, XML (*<oXygen/>*), Ant, visual programming, and more. Editors can be extended and share their resources with other plug-ins. One need not re-invent the wheel to add a small but useful contribution. To write your own editor, use the *Graphical Editor Framework* (GEF) plug-in.

There are plug-ins for team support (CVS synchronization, including SSH2 in Eclipse 3.0), JUnit, code instrumentation (*Clover*), Automated Software Quality (*Hyades*), web application server build and deployment (e.g., for *JBoss*, *Apache Tomcat*, *IBM Websphere*, *BEA Weblogic*), a platform delivery framework (*OSGi*), and more. There is an Eclipse project developing end-to-end web service development plug-ins.

Naturally, there is a *Plug-in Development Environment* (PDE), itself a plug-in, for creating plug-ins. And, for the JDT and the Eclipse IDE, the help system has a well-documented *Java Development Environment Plug-in Developer Guide* and a *Platform Plug-in Developer Guide*, respectively.

Eclipse on the Macintosh

So how is Eclipse integrated with the Mac? Eclipse uses the Carbon library via its *Standard Widget Toolkit* (SWT). While *Swing* generally implements its widgets with a combination of Java and native code, SWT maps its widgets to Carbon widgets via one-to-one C calls to the native widget APIs. When debugging, this means that when you run into a handle to a Carbon widget, you see precisely what you would expect if you were working in Objective C. By default, Eclipse obtains the Aqua look-and-feel and the best performance that Carbon can afford.

SWT emulates non-native widgets, and there is a separate SWT library for each platform. In addition, there's a shared library that glues SWT Java calls to the native API. At the lowest level, SWT maps to the Carbon Event Manager (**Listing 1**).

Listing 1: SWT Event Loop Life Cycle

Event loop using a window. Display interacts with the Carbon event manager to handle the application event loop via readAndDispatch(). A Shell is a window entirely handled by the Carbon Window Manager: it is instantiated with the Display as its parent.

```
Display display = new Display ();
```

Ruben Kleiman is a software architect and writer now developing an intelligent personal assistant. Formerly, he was Principal Scientist at Apple Computer, Inc., Architect at SGI, and Member of Technical Staff in AI at Mitre Corporation and the Microelectronics and Computer Technology Corporation. He has lectured on artificial intelligence and object-oriented systems, holds various patents, and has written for technical journals and popular magazines. Ruben lives in Northern California and he may be reached at ruben@rubenkleiman.com.

```
Shell window = new Shell(display);
window.open();
while (!window.isDisposed()) {
  if (!display.readAndDispatch())
      display.sleep();
}
display.dispose();
```

This is the main application thread. readAndDispatch() maps native events into SWT events. Carbon event loop timers are provided by Display methods. Listeners can be registered on any event type.

We can fork other threads for long-running tasks. Such threads can call back into the main thread, but only through an API that ensures that the interaction with the UI is safe. This is provided by Display's synchExec(Runnable) and aSynchExec(Runnable) methods; they allow the calling thread to block or not block, respectively, until the Runnable executes. For example, the forked thread may go off to zip some files for the user interface and, when done, use one of the *Exec(Runnable) methods to display a message to the user. From there on, Eclipse's user interface is based on SWT and JFace—a higher-level library providing *Model-View-Controller*-like access to SWT.

What about Cocoa? Cocoa and SWT are comparable frameworks, but SWT is founded on Carbon. The decision to use Carbon is arguable, but, since Carbon includes more lower-level APIs, it can provide some maneuvering room for the SWT implementors than Cocoa. There has been some work in pre-release versions of Eclipse 3.0 to support Carbon's HiView, but a higher priority of the Eclipse team seems to be to achieve a faster and more robust MacOS implementation in the near term. This, as we shall see, should be a welcome move.

How does Interface Builder fit in? Interface Builder generates code stubs suitable for Apple's Java extensions, but it generates NS* classes, not SWT or Swing classes. Since Interface Builder's output is a *nib* file, we could write a nib file reader in JavaCC and a generator for SWT or Swing using Carbon or Cocoa nibs. Alas, at the time of this writing, no such feature exists.

Finally: how usable is Eclipse on a Mac right now? The first version of Eclipse was released in November of 2001, but it wasn't until March 2003 that Eclipse support for the Mac started with Eclipse 2.1. There is little to complain about the Aqua look-and-feel (**Figure 1**). However, though on a Titanium PowerBook G4, at 667MHz and 512MB RAM, large project builds are as fast as one would expect, but tool tips and code assist are sluggish in Eclipse 2.1.2. Eclipse 2.1.2 is not up to par with its excellent performance on, e.g., Windows or Red Hat Linux. Eclipse developers are working to close this gap.

Installing Eclipse

Eclipse 2.1.x+ uses JRE 1.4+, so you need MacOS 10.2+. Eclipse requires 100 MB on disk and 256 MB of system RAM. To install Eclipse, go to http://www.eclipse.org and follow the download link to a MacOS version, with care to use Safari and the latest version of StuffIt, as instructed. We'll use Eclipse version 2.1.2 in this article. To uninstall Eclipse, just throw away the whole folder.

Getting Oriented

Three tips. First, the help system is indispensable. It has clear, often illustrated, step-by-step, task-oriented instructions, tutorials, APIs, and overviews of platform and plug-in features. The key areas you should peruse are the *Java Development User Guide* and the *Workbench User Guide*. Since almost all the information you'll ever need is there, it will pay you to get acquainted with these pages.

Second, select the **Help -> About Eclipse Platform** menu for an overview of the Eclipse features and plug-ins installed in your system. An Eclipse *feature* is a group of related plug-ins.

Finally, access your global preferences settings via the **Window -> Preferences** menu. Explore the **Workbench** and **Java** settings, especially those for the Java source editor's code formatting and code assist, as well as the auto-generation of comments and names for files, classes, methods, fields, etc.

What's Your Point Of View?

A View is a panel inside the Eclipse workbench window. For example, the Java Development Toolkit (JDT) plug-in and its extensions provide the following views:

- An editor view. There are editors supporting many file types (e.g., Java or AspectJ class, Ant build file, property file). Depending on your preferences, you also have access to other editors (e.g., Emacs) by clicking on a file and using the **Open With...** context menu. (Eclipse 3.0 early-access has an Emacs key binding option.)
- A view of the class inheritance tree (the **Hierarchy View**). In Eclipse 3.0 there is a **Method Call** view.
- The system console (the **Console View**).
- The "todos," error, build and other tool warnings, etc. (the **Tasks View**).
- The hierarchical view of all of the resources in your project (the **Package Explorer View**).
- An outline of a class's members and imported classes (the **Outline View**).
- **Packages**, **Types** and **Members** views that display and let you to focus on Java packages, classes, and members, respectively.
- The **Synchronize** view allows you to synchronize your project with a CVS repository.
- If you use the Apache Ant tool, the **Ant View** allows you to look at your Ant xml file's tasks in outline and run them individually, or to look at the properties of your Ant file.
- Eclipse's *AspectJ* IDE plug-in enhances most of the above views to support Aspect-Oriented Programming (AOP) using the AspectJ language. For example, the **Outline** view lets you

see whether a member is advised, and by whom and how it is intercepted.

Many more views are available from free and commercial plugins. Views collaborate through the plug-in architecture. For example, the **Members** view's contents on which package is selected in the **Packages** view. Each view has a mini-toolbar, and sometimes drop-down and contextual menus, as well as tool icons. Try them out. Double-clicking on a view's mini-toolbar expands/minimizes the view.

Although you can open any of these views via the **Window -> Show View** menu, you will typically work with a group of views: this is where perspectives come in.

It Depends On Your Perspective

A perspective is just a group and placement of views in a window (**Figure 1**). The *Eclipse Java Development Toolkit* plug-in comes with some ready-made perspectives. You can create your custom perspectives based on existing perspectives by combining and arranging the views you want.

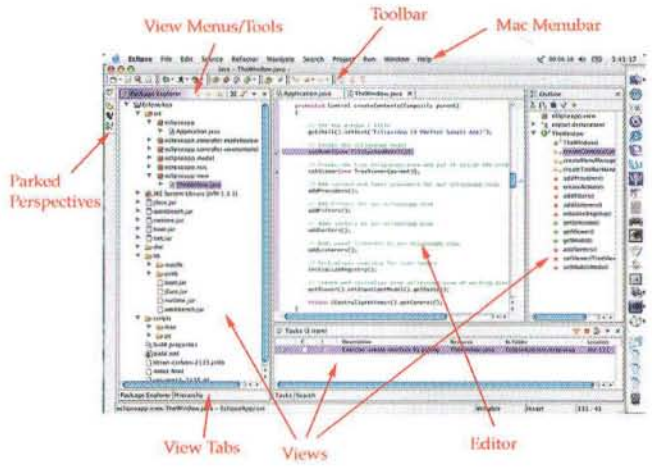


Figure 1. *Default "Java Perspective"*

JDT's most useful perspectives are the so-called Java, Java Browsing, and Debug perspectives. You can open them by selecting the **Window -> Open Perspective** menu. You create new perspectives by adding, deleting, and dragging views around an existing perspective. When you place more than one view on the same real estate, view selection tabs appear. You can open multiple perspectives on separate windows.

Creating A Java Project

Let's create a project. (If you want to learn more about what follows, review the help system's **Java Development User Guide -> Getting Started -> Basic Tutorial** section.)

- Open the Eclipse IDE.
- Go to **File -> New -> Project...** The project wizard's dialog lets you select the type of project. Select **Java Project** and click **Next**.
- Name the project EclipseApp and specify its location in disk (the default location is the workspace folder). Click **Next**.
- This **Java Settings** dialog allows you to specify the directories for the source and class files and libraries. First, make sure that you are looking at the dialog's Source tab. The default directory for sources should be EclipseApp/src. If it is not, select the EclipseApp folder icon, click **Edit...**and then click on the **Create New Folder...**button and name the folder src. Ensure that EclipseApp/build is the **Default output folder** at the dialog's bottom. If it is not, type it in.
- Click on the **Libraries** tab to set up our build path. You should see your default JRE or JDK already in it. We won't need to add any libraries besides the JDK's, so click **Finish**.

You should now see the EclipseApp project in the **Package Explorer** view of Eclipse.

Working With Source Code

In the **Package Explorer** select the src folder and select **File -> New -> Package**. In the dialog, name the package mypackage and click **Finish**. Now select **File -> New -> Class**. Fill in the form to create a class named HairyClass in mypackage. The class opens in the Java editor. Notice that the stub for the constructor is marked with a // TODO task. The latter appears in the **Tasks** view; double-click on a task to go to its location. You can set task preferences in **Windows -> Preferences -> Task Tags**.

For the final contents of this class, see **Listing 2**. Let's enter the public field at the top. Just type pu and enter **Command-Space**. The code assist feature brings up the available completions in the context. You can navigate the completions menu with the arrow keys or mouse to view Javadocs, if available, or keep typing to reduce the suggestions. Hit return to fire a highlighted completion. Type the rest of the line using autocompletion to get the hang of it. When you get to third field, type up to public Ve and use autocompletion to select Vector. Note that java.util.Vector is now in the imports list. Now delete this line, retype it without autocompletion, and save it. Saving a file recompiles the class. There should now be squiggly lines around the name Vector. When the mouse hovers over the squiggles, the error message displays. There should be an error icon on the left margin of the line: depress the mouse on it to get suggestions for fixing the problem. Double-clicking on any suggestion makes Eclipse implement it. As you've been navigating the code, you've probably tried the Outline view, or perhaps you've switched to the Java Browsing perspective or created your own. Finish up this class.

Now create a class called MyMain in the same package and, in the **New Java Class** dialog check the box to create a main(String[] args) stub. Enter the code in **Listing 2.**

HairyClass is rather, well... hairy? Select the field identifier foo and then the **Refactor -> Rename**... menu. Rename the field to myVector. You can use the **Preview**... button to inspect the cascade of changes resulting from this; e.g., in MyMain, the reference to foo has changed to myVector. What's wrong with this picture? Select the myVector field name in HairyClass and **Refactor -> Encapsulate Field**... In the dialog that appears, make sure that **keep field reference** is unchecked and click **OK**. If you now inspect MyMain, you should see the field access changed to a method access and myVector is now private. Now proud of HairyClass, we rename it GreatClass using the same refactoring feature by selecting the class name or the class icon in the **Package Explorer**. Let's go further by creating an interface for it. Select the GreatClass name and **Refactor -> Extract Interface**... Name the interface IGreatClass and check off all methods *except* setMyVector(Vector). Note that GreatClass implements IGreatClass. Now let's be silly and delete getMyVector() from GreatClass and save. An error appears at the top: click for suggestions and ask it to **Add unimplemented methods**. Now getMyVector() is re-instated, with a pointer to the method's Javadocs in IGreatClass. For fun, inline the execute method.

Refactoring is a subject in itself. Explore the refactoring features in Eclipse and you'll see that it is very hard to break them. You can rename or move anything, with collateral refactoring of comments and literals as well, if you want. It is fun to use the refactoring dialog's **Preview**... button to try lots of what-ifs with your code. For more on refactoring, see the help system's **Java Development User Guide -> Tasks -> Refactoring**.

Listing 2: HairyClass and MyMain

Original listings of HairyClass and MyMain.

```
Class HairyClass
{
  public Vector foo;
  public int count;

  public void execute()
  {
    foo = new Vector(count);
  }
}

Class MyMain
{
  public static void main(String[] args)
  {
    HairyClass hc = new HairyClass();
    hc.count = 10;
    hc.execute();

    if(hc.foo.size() == 10)
      System.out.println("Hairy class, but it worked!");
  }
}
```

Builds

By default, the project compiles incrementally as you save files. There are also tools for generating Javadocs, JARs, adding your own "external" tools, and so on. Of course, for complex projects, the natural tools are Apache's Ant or Maven.

Eclipse has an installed Ant plug-in that provides an Ant editor, two views, and a dialog. Both views are outliners: one is used to navigate elements in the editor; the other to quickly execute any target. The Ant dialog lets you configure your build and change Ant options. Also, there's a trend to generate Ant files rather than execute black-box tools. For example, Eclipse's Javadoc generator (**File -> Export -> Javadocs**) will create an Ant file for you that runs Javadoc with all your settings. This lets you include the Javadoc (or any other tool's) Ant output into your main Ant.

Finally, with Eclipse's *AspectJ* IDE plugin, there is support for multiple build configurations, allowing you to, e.g., switch between pre-production and production time builds.

Running And Debugging

Each project can be assigned multiple configurations for launching and debugging. The debugger provides all of the basic features, including remote JVM debugging. This makes it easy to work with web application servers; and, there are IDE plugins for JBoss, Weblogic, Websphere, and Tomcat to control the server. For illustrated, step-by-step instructions for running and debugging your project, start with Help's **Java Development User Guide -> Getting Started -> Basic Tutorial** section.

Testing And QA

Eclipse's JUnit plug-in helps you to set up test cases by mitigating the tedium of manually creating test classes. There are wizards to create test cases and suites, run them, create test configurations and debug them. If you are interested in this, step-by-step instructions are available in the help system's **Java Development User Guide -> Getting Started -> Basic Tutorial**.

To understand how well your tests cover the possible runtime cases, you can instrument your code with the *Clover* plug-in (http://www.thecortex.net/clover). *Clover* also gathers statistics to analyze your code's runtime characteristics.

Finally, Eclipse's *Hyades* plug-in is an ambitious project to provide a reference standard for tracing, testing and monitoring; it implements the UML2 Test standard. It includes user interfaces for profiling, testing (from test case modeling to implementation), logging, and platform-specific data collection. *Hyades* provides modules for cross-platform and multi-platform development. Stable versions of *Hyades* are available from the Eclipse website.

Teamwork

If you are one of those rare individuals in a development team, you'll use the **Synchronize** view or, for a full workbench, the **CVS Repository Exploring** perspective. You can get to the latter via **Window -> Open Perspective -> Other**... Set up and creation of projects is straightforward. Eclipse 2.x supports SSH1, and SSH2 will be supported in Eclipse 3.0 (in early release at the time of this writing).

Conclusion

Though we've only scratched the surface of the Eclipse IDE, by now you should have an insight into Eclipse, why you might want to use it, and how it fits into the Macintosh universe. Future articles will focus on cross-platform development using Eclipse's JFace and SWT libraries, on Objective C development, and on more plug-ins.

References

Shavor, S., D'Anjou, J., et. al. *The Java Developer's Guide to Eclipse*, Addison-Wesley, 2003; 896 pp.

Gamma, E., Beck, K., *Contributing to Eclipse: Principles, Patterns, and Plugins*, Addison-Wesley, 2003; 320 pp.

Budinsky, F., Steinberg, D., et. al., *Eclipse Modeling Framework*, Addison-Wesley, 2003; 704 pp.

Gallardo, D., Burnette, E., McGovern, R., *Eclipse in Action: A Guide for the Java Developer*, Manning, 2003; 380pp.

By Dave Wooldridge

Communicating with Customers

How the New CAN-SPAM Act Affects Software Developers

Over the last few months, you've undoubtedly heard talk of the new CAN-SPAM Act that became federal law in the United States on January 1, 2004. The purpose of the CAN-SPAM Act is to help regulate and reduce the onslaught of spam that plagues all of our inboxes, but the perceived effectiveness of the new law has been met with mixed reactions. Many people feel that the law is not strict enough and actually legitimizes unsolicited e-mails. Still receiving hundreds of spam messages per day, touting organ enlargement solutions, get rich quick pyramid schemes, generic prescription painkillers and Viagra alternatives? You're not alone.

Whether you believe in the validity of the CAN-SPAM Act is irrelevant. Violations carry a severe penalty. Depending on the specific offense, penalties include jail sentences of one to five years, plus hefty monetary fines. CAN-SPAM also discusses the offering of a bounty of at least 20% of the collected fines to citizens who report violators. And with the strong hatred the general public has for spam, you can count on seeing a lot of people submitting reports, especially if they can make money for putting spammers behind bars! As a federal law in the U.S., it supercedes any regional state anti-spam legislation that currently exists, although some states may still pursue their own prosecution of spam offenders. Needless to say, anti-spam laws should not be taken lightly. To read the entire CAN-SPAM Act in its entirety, visit http://www.spamlaws.com/federal/108s877.html

For those of you who find legal documents to read more like alien Klingon scripture than actual English can rest easy. This month's column will summarize the key points of the CAN-SPAM Act and how they directly affect software developers. As a programmer communicating with only your existing customers and opt-in e-newsletter subscribers, you may wonder why you should be concerned with this new law, especially since many of the rules apply to unsolicited e-mail. The simple truth is that people lead busy lives and while they may have voluntarily subscribed to your e-newsletter last October, come January they may not have any recollection of doing so when they receive your latest e-mail. They report it as spam, not remembering they actually did subscribe at some point, and then before you know it, you've got men in black suits with badges knocking on your front door. You can never be too careful or conscientious when it comes to people's e-mail privacy. Even if you employ double opt-in procedures for your e-newsletter and customer mailing lists, it is important for you to understand the law. Honor the new rules with every bulk e-mail campaign you deliver, so that you do not accidentally wind up on the wrong side of a lawsuit. Plus, for those of you who do maintain a regular e-newsletter mailing list (and if you don't, you should), we'll explore some effective techniques for maintaining your subscriber base, strengthening customer loyalty and increasing software sales.

The "From" Address

The "From" line should state your name or your business name with a valid e-mail address. Never forge a false e-mail address and never substitute the name with marketing jargon (like "Special Offers").

Besides the fact that it is against the law, using a false or different e-mail address other than the one you are sending from is a sure-fire way to get caught by spam filters. For example, you decide to send the latest e-newsletter to your mailing list over the weekend from home, using your personal DSL account, but you want the e-mail to look professional, so you forge the "From" line to be your company e-mail address. While this may seem harmless enough, two problems can arise. One, your ISP's mail server may detect the false "From" header and block the bulk e-mail from being sent. And two, many of the anti-spam filters that people use are smart enough to spot false "From" headers and automatically send your e-newsletter to the "Junk Mail" folder where it gets deleted. You don't want to spend valuable time and money producing an e-newsletter that never gets read.

The "Reply-To" Address

Always use a "Reply-To" e-mail address that is guaranteed to be valid and active for at least thirty (30) days after you send the e-mail.

Dave Wooldridge is the founder of Electric Butterfly (`www.ebutterfly.com`), the web design and software company responsible for HelpLogic, Stimulus, UniHelp, and the popular developer site, `RBGarage.com`.

For those recipients who decide they want to unsubscribe, they often do not scroll down far enough to see the unsubscribe instructions at the bottom of the message, so they will click the Reply button and e-mail you a request to be removed from the list. Manually processing these kinds of unsubscribe requests can be tedious and time-consuming, so an effective way to deal with this problem is to set-up the "Reply-To" e-mail address with an auto-responder message. Most web hosting providers and ISPs support an e-mail auto-responder option, which shoots back a pre-defined message to any person who e-mails that address. Most companies use them for notifying clients when specific employees are on vacation, etc. but they can also be used to provide customer service to your mailing list subscribers. Set up a new e-mail address such as replies@yourcompany.com that will not be used for any other purpose than as your e-newsletter's "Reply-To" address. Add an auto-responder message to that e-mail account. The message should thank the sender for contacting your company and explain politely that in order to unsubscribe from the mailing list, they should visit the following link (and then include the unsubscribe URL). It's also a good idea to list the URL or e-mail address for contacting customer support in case the sender wanted to ask a question or receive technical assistance on one of your software products. Make sure your auto-responder is in plain text format to ensure that even text-only e-mail programs can properly display the message.

The "Subject" Line

Your e-mail's "Subject" line should be descriptive and directly related to the content of your message. Using misleading or deceptive "Subject" lines (such as "Your account has been declined!" for a home loans promotion) can carry hefty penalties.

Using vague "Subject" lines may seem like a brilliant ploy to get curious recipients to open your e-mail message, but in today's world of sophisticated spam filters and over-cautious users, e-mails such as "Check this out!" or "Deal of the Century!" will, more often than not, end up unread in the "Junk Mail" folder. So not only are they against the law, but these questionable "Subject" lines will not stand out in an already oversaturated sea of vague spam headlines.

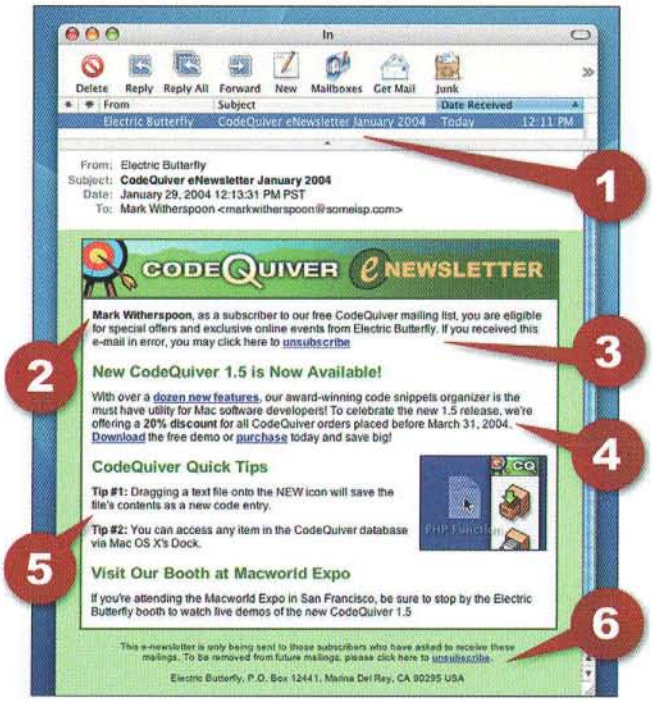


Figure 1. *Even though recipients voluntarily subscribed to your opt-in mailing list, take extra care to ensure that your e-mail messages are effective, while remaining compliant with state and federal laws.*

Your goal is to create a subject line that catches the attention of recipients while honoring the true content of your e-mail message. Including your company name in the "Subject" line may prove effective, but unless you're a well-known entity like Adobe or Macromedia, people may not instantly recognize your name. For small software companies and independent shareware developers, the eye-catching word that will jog users' memories is most likely the name of your product. Using our fictional software product, CodeQuiver, as an example, the "Subject" line in **Figure 1, Item 1** is "CodeQuiver eNewsletter January 2004." You could obviously make the line longer with additional words such as "Electric Butterfly January 2004 eNewsletter: CodeQuiver Update," but you run the risk that users' have their incoming e-mails listed with a narrow width for the "Subject" field, displaying only the first part of the phrase: "Electric Butterfly January 2004 eNewslet..." By placing the identifying word "CodeQuiver" in the beginning of the phrase, "CodeQuiver" will always be visible, no matter how narrow the "Subject" field is set in email programs. Our "Subject" line will now prove to be much more eye-catching for those users who quickly recognize the product name, but not our company name.

But what if your e-newsletter promotes more than one software product? Due to space limitations, you don't want to list all your products in the "Subject" line. If you have several software titles and they cater to different markets (some are consumer-oriented, some are developer tools, etc.) then you may want to consider sending out several e-newsletters with each one targeted to specific subscribers in your mailing list. Users of your consumer-based products may not want to read in-depth news about your developer tool offerings. The most effective e-newsletters are those that are brief and focused on the interests of the recipient. Less is more. They will remain subscribers as long as the content remains useful and is easy to read quickly.

Apple is a good example of a company that utilizes e-newsletter targeting. When signing up online to receive e-mail news from Apple, you are asked to select the specific e-newsletters that interest you (usually represented by a list of checkboxes). "Apple eNews" announces the latest updates, special offers and user tips for Mac OS X, iLife, and other consumer-based Apple products. "QuickTime News" showcases multimedia authoring, special promotions on Final Cut Pro, Final Cut Express, and other QuickTime-based software. "New Music Tuesdays" provides the latest music news and releases available in Apple's iTunes Music Store. "ADC News" is the official e-newsletter for Apple Developer Connection, focusing on Xcode, Cocoa, and third-party developer tools. If Apple combined all of these different topics into one consolidated newsletter, it would weaken their effectiveness to generate sales. There would be too much information in one e-mail with much of the content reaching the wrong people whose interests lie in other areas. Let's say a proud father who creates iMovie videos of his daughter's soccer games is interested only in hearing about the latest iLife and QuickTime news. If he received a general, "all-in-one" Apple e-newsletter that was packed with lots of unrelated articles on Objective-C programming techniques, he may grow impatient wading through the content just to find the information that suits his needs. If he unsubscribes after a few issues, Apple then loses the ability to tell him about new products and special offers that he *would* be interested in, losing a potential sale. By subscribing to the targeted e-newsletters "Apple eNews" and "QuickTime News," he only receives news related to the topics he wants to read about, providing a consistent and reliable line of communication from Apple to his e-mail inbox.

If you don't have a way to configure and manage targeted groups within a single mailing list on your own web server, there are several third-party services such as Microsoft bCentral's ListBuilder (http://www.listbuilder.com/) which provide these advanced features and more. ListBuilder generates a sign-up web form that enables subscribers to select their topics of interest (which ListBuilder calls "Groups") from a list that you pre-define. ListBuilder hosts your mailing list database and automatically manages all submissions and unsubscribe requests. Using ListBuilder's convenient web admin tools, you can elect to send an e-newsletter to either the entire list or to only specific Groups.

E-mail Identifiers

If you e-mail an unsolicited message promoting your products or services, you must identify the e-mail as an advertisement in the "Subject" line. Although there is no predefined standard, most e-marketers use the label [ADV]. Make sure the label is easy to spot by placing it at the beginning of the "Subject" field. For example, "CodeQuiver Improves Programming Productivity!" would become "[ADV] CodeQuiver Improves Programming Productivity!"

Solicited e-mails that are delivered to opt-in subscribers are not required to include any kind of identifier, but since some recipients may accidentally forget that they did subscribe voluntarily, you can avoid false spam accusations by politely reminding them in the first paragraph of your message. Personalize the e-mail with the recipient's name (**Figure 1, Item 2**) to show that it's not a generic message, but that it's addressed specifically to them. Then add a brief sentence that reminds them that they *chose* to receive your e-mails (**Figure 1, Item 3**). This can be done in a very elegant way by mixing the phrase with some dialog that explains why it is beneficial for them to *continue* receiving the e-mail messages. Using the fictional CodeQuiver e-newsletter as an example, we accomplished both goals with one sentence by saying: "As a subscriber to our free CodeQuiver mailing list, you are eligible for special offers and exclusive online events from Electric Butterfly."

Any e-mails (both solicited and unsolicited) that include sexually oriented material must be labeled as such in the "Subject" line. Most software developers won't have a need for this kind of labeling, but just in case some of you develop adult multimedia applications or web sites, it might be helpful to note that many adult magazines use the identifier [NUDITY] in the "Subject" line when e-mailing advertisements to their subscribers. Some states in the U.S. consider the unsolicited delivery of sexually oriented e-mails to be against the law and enforce serious penalties, so it is strongly recommended to consult an attorney before sending e-mails with this kind of content.

Unsubscribe Requests

Whether you are sending solicited or unsolicited e-mails, each and every e-mail is required to include an easy to find "unsubscribe" link somewhere in the body of the message. A recipient should be able to opt-out from your mailing list in one step (or no more than two steps). Your opt-out instructions should be very easy to follow, making the process extremely simple. Forcing recipients to jump through a complicated string of steps as a deterrent to keep them from unsubscribing is not only against the law, but will only serve to agitate potential customers. All unsubscribe requests must be honored within ten (10) days of receiving them.

Most e-marketers place the "unsubscribe" link at the bottom of the e-mail message. For opt-in e-newsletters, this is a good place to once again remind recipients why they are receiving the e-mail. In **Figure 1, Item 6**, the "unsubscribe" link is accompanied by a brief sentence of instructions: "This e-newsletter is only being sent to those subscribers who have asked to receive these mailings. To be removed from future mailings, click here to unsubscribe". Just in case some recipients do not think to scroll down to the bottom of the message, placing a second "unsubscribe" link near the top of the message provides added convenience. A subtle way to do this without cluttering your content is shown in **Figure 1, Item 3**, where the "unsubscribe" link is also added to the tail-end of the introductory paragraph. This may seem like overkill, but if you only include the one "unsubscribe" link at the very bottom, you'll be surprised by how many e-mails you receive from people who cannot figure out how to opt-out.

Another way to prevent confusion is to provide a confirmation page that informs people that their unsubscribe request was successful and that they should not receive any more mailings from you. If your mailing list system requires subscribers to opt-out by replying via e-mail with the word "unsubscribe" in the "Subject" line, use an auto-responder to send an e-mail message back to the user, acknowledging their request.

It is your responsibility to ensure your automated opt-out process is working properly. Double and even triple-check your opt-out feature. If people are unable to successfully unsubscribe from your mailing list, their first guess will not be a faulty system. They will assume the worst: that you are trying to take advantage of their good will. Feeling frustrated and exploited, these potential customers just turned into angry enemies who may falsely report you as a spammer.

A Physical Postal Address

All solicited and unsolicited e-mails are required to include the valid postal address of the sender. This means your full street address, city, state/province, zip code, and country. It's standard practice to place the postal address at the very bottom of the e-mail message, beneath the unsubscribe instructions.

If you are a home-based shareware developer and would rather not give out your home address (for fear of customers knocking on your front door on a Saturday afternoon), then rent a P.O. box from your local post office or mail supplies store. Prices are very affordable, typically ranging from $40 to $100 (US) per year. Using a P.O. box as your public company address will also shield your home residence from receiving junk mail from address collectors.

Guilty By Association

The CAN-SPAM Act also bans automated e-mail harvesting, so programming a robot script to crawl web sites to collect e-mail addresses is strictly prohibited. This means that purchasing a list of harvested e-mail addresses from a third-party company or utilizing a bulk e-mail service whose database consists of

harvested e-mail addresses is also against the law.

Not only are you responsible for your own e-mail practices, but you are also responsible for all affiliates, resellers, and marketing agencies who send out e-mails on your behalf. Whether you have directly instructed these partners to promote your products/services or if they are acting independently in the hopes of increasing their own commissions, you are ultimately responsible for their actions since the e-mails are tied directly to the promotion of your software company. This is usually an easy element to control with resellers and marketing firms, but it can become staggeringly difficult to manage with an affiliate program. Enticing web site owners to promote your software by using special purchase URLs that track and reward commissions based on sales they generate can be an extremely powerful marketing tool. Affiliate programs like this have proven to be incredibly successful for online retailers like Amazon.com, but managing the selling tactics of these affiliate members can be somewhat tricky. Since affiliates make money when customers buy your software through their special affiliate hyperlinks, they often barrage the public with web advertisements and unsolicited e-mail promotions in the hopes of increasing their commissions. To enforce that all affiliates obey local and federal spam laws, all program members should be required to sign a Terms & Conditions document that limits your liability and states their selling boundaries. Let it be known in no uncertain terms that any affiliates who violate the rules will have their membership terminated and may face legal repercussions.

Extra Measures

Beyond the new CAN-SPAM Act, there are several things you can do to further safeguard your e-newsletter campaigns and make them more effective as a key communication platform between you and your subscribers.

Double Opt-in. Don't let people clutter your mailing list with invalid e-mail addresses or the unauthorized e-mail addresses of others. After signing up online, a verification message should be sent to the submitted e-mail address, asking the owner to confirm their subscription request by clicking the included URL. If the request was initially made in error, they can opt-out by simply doing nothing. If the submitted e-mail address is invalid, then it won't be added to the mailing list (since there is no one on the receiving end to confirm the request). Making the opt-in path a simple two-step process will ensure that your subscribers are genuinely interested in receiving your e-newsletters.

Privacy Policy. Every web site and mailing list should have a privacy policy. Many people are so afraid of ending up on unsolicited spam lists that they refuse to submit their e-mail address in any online form unless you tell them upfront that their e-mail address will never be sold or distributed to any one outside of your company. On the same screen as your e-news web form, include a brief sentence guaranteeing that you will never share their personal information and include a URL link to your privacy policy. Some e-marketers even include the "privacy policy" URL at the bottom of every e-mail (in the same paragraph as the "unsubscribe" link).

Never Use Attachments. With so many computer viruses being distributed through e-mail, the general public has become very wary of e-mail attachments. The fear has escalated to the point where e-mails with attachments that were not sent by a friend or family member are usually quickly deleted without being opened. If you're sending an HTML e-newsletter, never send the HTML document as an attachment – always embed the HTML into the main body of the e-mail. And never send the web graphics used in the HTML e-newsletter as attachments. Aside from the fear factor of attachments, using relative paths to attached graphics in your <IMG> tags does not work properly in some e-mail programs, incorrectly displaying your HTML e-newsletter with broken images. The professional way to display graphics within your HTML e-mails is to host the graphics on your web server and then use absolute URLs in your <IMG> tags. For example, instead of using a relative path:

```
<img src="logo.jpg" width="300" height="50" border="0">
```

use an absolute path:

```
<img src="http://www.yoursite.com/logo.jpg" width="300" height="50" border="0">
```

Never Use HTML Forms. Avoid embedding forms in your HTML e-mails. Besides the fact that some older e-mail programs do not support HTML forms, they tend to scare or intimidate recipients. People are often suspicious of what is being activated when the "Submit" button is clicked and where their data is being sent. Even if your subscribers trust your company, they may not trust the e-mail since they have no way to prove its authenticity. The last couple years have seen hundreds of e-mail scams disguising themselves as eBay or PayPal, asking customers to verify their passwords, bank account numbers, social security numbers, etc. through e-mail-based forms. Not wanting to be the next victim, subscribers may opt to simply delete the e-mail or worse yet, unsubscribe. If you want e-mail recipients to participate in an online poll or survey, post the HTML form on your web site and then promote it in your e-mail with a URL link to that form page.

Useful Information. Don't assume people will remember what your software does by only promoting the latest release or sale price in your e-newsletters. In **Figure 1, Item 4**, we promote the new CodeQuiver release while briefly reminding readers of CodeQuiver's primary function. And if your e-newsletters only encourage software purchases, then their usefulness may diminish in the eyes of readers. By providing convenient user tips and techniques (**Figure 1, Item 5**),

recipients will be motivated to remain an active subscriber for a much longer period of time.

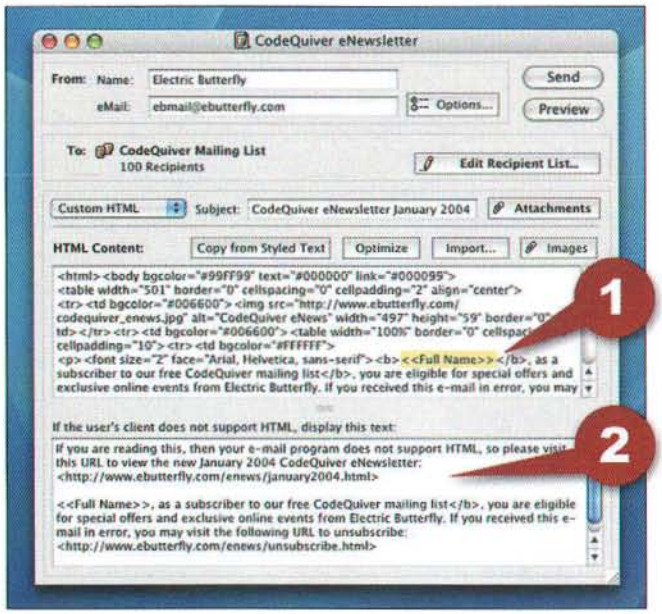


Figure 2. *Using an e-mail merge program like IntelliMerge, you can send personalized e-mails to your customer mailing list. Sending your e-mail with a multi-part MIME type will allow your message to display on both HTML-enabled and text-only e-mail programs.*

Personalize. If you decide not to use a mailing list service such as Microsoft bCentral's ListBuilder or custom web server software, but elect to e-mail your mailing list from your own computer, you should still personalize each e-mail with individual names and e-mail addresses (**Figure 1, Item 2**). Personalizing your e-newsletters and customer communications adds a nice touch, building a direct bridge between you and each recipient. Do not send one message with your entire mailing list hidden in the BCC field. Many ISPs will refuse delivery, auto-detecting these kinds of bulk e-mails as spam, and even if they do get past ISP gateways, most personal spam filters will discard them. The best way to send out your own e-newsletter is by sending it to each recipient individually as a separate, personalized e-mail. Manually, this would take far too much time, but there are several mail merge software applications available for the Mac that can help automate this process. Intelli Innovations' IntelliMerge (http://www.intellisw.com/intellimerge/) is one such solution. Import your mailing list into IntelliMerge and then embed special database tags in your e-newsletter's text to dynamically populate those tags with the appropriate subscriber information when e-mailed via IntelliMerge. For example, the <<Full Name>> tag (**Figure 2, Item 1**) will be replaced with the subscriber's name when IntelliMerge delivers the e-mail (**Figure 1, Item 2**). If you prefer to use Mac OS X's Mail and Address Book, then Christian Fries' SerialMail (http://www.christian-fries.de/osx/SerialMail/) may be a viable solution for you.

Catering to Text-Only E-mail Programs. Due to personal preferences, limited access to the latest e-mail software, and/or the ongoing threat of e-mail-driven viruses, some people are unable to read HTML e-mails or choose to disable the HTML option. Since these precious few may be either existing customers or potential new customers, don't exclude them from enjoying your e-newsletters. This doesn't mean you have to design text-only e-mails to meet the lowest common denominator. You can configure the e-mail with a multi-part MIME type. This enables one part of the message to be designated as HTML, while the other part is relegated to plain text. Setting the current message in IntelliMerge as "Custom HTML" turns it into a "multi-part" e-mail, displaying two fields: one for the HTML version of your e-newsletter and one for the plain text version. In HTML-compatible e-mail programs, the HTML version will automatically display, leaving the plain text version hidden. In text-only e-mail programs, the plain text version will be displayed by default. If you prefer everyone to read the more elegantly designed HTML version, you can post it as a web page on your site and include the URL and viewing instructions in the plain text section (**Figure 2, Item 2**), so that text-only recipients can view the HTML e-newsletter in their web browsers.

Stay Informed

Before sending out bulk e-mails from your own computer or web server, review your ISP or web hosting provider's e-mail policies. E-mailing thousands or even hundreds of people at one time can set off "red flags" with your ISP and web hosting provider. You don't want to send out your latest software announcement and then have your account shut down for violating an existing e-mail policy. Internet access providers are becoming increasingly strict in an attempt to control and reduce spam. Although you may only be delivering an e-newsletter to your double opt-in list of happy, legitimate subscribers, your ISP is not aware of these details – all they see are traffic logs of 3,000 e-mails being sent from your account within the short span of a few minutes. Using a third-party service like Microsoft bCentral's ListBuilder avoids these kinds of complications.

Stay on top of the latest local and federal anti-spam developments. Even though your opt-in e-newsletter mailing list would not be considered spam, it's a good idea to be aware of any new changes or amendments, so that your e-mails are always compliant with the law. When in doubt, consult an attorney so as to choose the right e-mail strategy that best protects you and your company. Better safe than sorry in the precarious world of e-mail marketing.

By Jono Bacon

Qt

A multi-platform graphical toolkit

The birth of the toolkit

In the software development world, there are many tools and services available to help developers maximise their application development cycle, and squeeze every bit of functionality out of their products. These tools include compilers, debuggers, development environments, profilers, and, of course, graphical toolkits. In this article, I will review a toolkit that is rapidly growing in reputation and ability; Qt, the flagship product from Norwegian company Trolltech. They state that Qt can make the "Code Less. Build More. Compile Anywhere" motto a reality.

In this article I will be reviewing the latest version of Qt available at the time of writing, version 3.2.2. I will assume you have never used Qt before, and so will evaluate the different aspects of the Qt system. Although I will be discussing many different features, not all of them are specific to the new 3.2.2 version; I will discuss specific features in this new version towards the end of the article. This way those of you familiar with Qt can find out what is new in 3.2.2, and those not can get a perspective on the whole Qt toolkit, and its features.

It's all about the code

Qt is a multi-platform toolkit. "Nothing new there" I hear you scream, citing Java as an example, but Qt offers a slightly different method of creating graphical applications. First, Qt is a native toolkit. This means that when you create a Qt application and compile it, a native binary is created to run on each specific operating system. The speed implications for the application are therefore drastically improved, and large applications should work, theoretically, as fast as any other native software, with the added benefits of Qt's cross platform source code.

The actual cross platform nature of Qt is largely a feature of its Application Programming Interface (API). The concept here is that the source code remains the same for each platform version of Qt, so to create a binary for another platform, you just recompile using the Qt for that platform. Sounds great in theory, but does it work? We will investigate this later in the article. Before we get onto the cross platform nature of Qt, however, let us first look at what Qt can do and what is available in it.

Qt Features

I have been familiar with Qt as a product since the release of the first version, and each major release has put more and more functionality under the hood. The first and most basic set of functionality is for creating graphical interfaces. Qt includes widgets (the equivalent of Controls) for buttons, checkboxes, radio buttons, tabs, icon panes, canvases, dialog boxes, and other common interface elements. In addition to these elements, there are special dialog boxes (such as a file picker, about box, etc.) and facilities that save the developer from having to re-implement these functions over and over.

In addition to these graphical components, Qt includes a number of additional features. One of the most critical set of features are the convenience classes for handling data and types. Qt provides a number of these classes to support Arrays, Strings, Vectors, Maps, and many more types. In addition to these basic classes there are also classes to handle networking, sockets, file transfer, sound, and many other aspects of software development. It is good to see that Trolltech has created a rich API that not only provides graphical widgets, but also provides a number classes that make the day to day tasks of programming it a little bit easier. Another feature in Qt is the concept of additional modules. These extra modules have specific functionality that can be added to the API. These modules include a graphical canvas, database access, networking, OpenGL, and XML facilities.

Although Qt is primarily a graphical toolkit and convenience classes, Trolltech has worked to supplement this API with tools and facilities to assist in the development of projects. These additional tools include the graphical dialog box creation tool Qt Designer, the translation tool Qt Linguist, the documentation tool Qt Assistant, and the compilation tool QMake. We will look at each of these tools later in this review.

Qt Usage

Qt is a C++ based toolkit, and Object Orientated Programming (OOP) is fundamental to using Qt. Each of the

Jono Bacon is a writer, musician, and developer based in the United Kingdom. Jono has an avid interest in Linux/Open Source and has been involved with a number of Open Source projects.

components is available as a class (such as a QPushButton to create a push button widget), and each class has a number of methods to handle common tasks and features. Although some developers use procedural programming for graphical applications, the nature of OOP lends itself well to GUI programming due to the fact that inheritance is fundamental to GUI's. As an example, there is the general concept of a button (something that you click on), and then specific types of buttons (push, radio, toolbar, etc). In Qt, inheritance is used in this way so that the general QButton class is inherited by a more specific QPushButton class, for example. This process gives the developer all the functionality that could be needed for that specific widget, and the lower level functionality for its inherited class. This is incredibly flexible and is implemented well in Qt.

Development of applications can be quite varied in Qt. When developers begin programming with a toolkit, they often use classes with a lot of functionality that is rarely used, that does nothing but increase the size of the binary. In Qt things are a little different. There are different classes that serve different uses. As an example, there is a QMainWindow class that provides a lot of functionality for typical office type applications with menus, toolbars, and a main content area. Although great for this kind of functionality, it may seem a little bit of overkill for more simplistic applications or graphical elements. Say you wanted to create a simple window with a single text box to type your password into, you could use a QDialog class that is much more efficient.

One of the most interesting features of Qt is the way in which user interaction is handled. In many toolkits there is the concept of events, messages, call backs, etc., where a particular widget will start a particular event when the user does something. It is then the programmers responsibility to capture the event and do something constructive in response to it. Trolltech has taken this (often bizarre and complicated) concept and refined it, coming up with a solution named Signals and Slots. The basic idea is that each class has a number of pre-defined signals that are emitted when something happens, such as clicking on a button or selecting an item in a menu.

For example, the QPushButton class that is used to create a simple push button (such as an OK and Cancel button in a dialog box) has a clicked() signal; this signal is emitted when someone clicks on the button. This particular signal can then be connected to a slot, which is any normal function. Using this system, you can easily connect any function to a user interaction. It requires only a single line of code to perform this connection.

Additional Tools

Earlier I mentioned that there are some tools included with Qt that can assist in developing your applications. These tools come in the form of Qt Designer, Linguist, Assistant, and QMake. All of these tools are genuinely useful, and speed up development with Qt.

Qt Designer

Qt Designer essentially gives you the ability to draw your graphical interface visually by dropping interface elements onto a window. Qt Designer is a very flexible tool, and has support for all of the graphical widgets that are available in the toolkit. Not only can you add items such as buttons, checkboxes, radio buttons, scrollbars, textboxes, etc., but you can also add menus, and their items. Adding the components to your application window is as simple as selecting the widget from the toolbar and then drawing it. Qt Designer does not stop there in terms of creating your interface. It also gives you the ability to define your signal/slot connections. This procedure is started by selecting the object that will emit the signal, then entering the name of the slot that the signal connects to.

When an interface has been created and the file is saved, Qt Designer will store your interface in a .ui file that is comprised of special XML code. It is a wise move on the part of Trolltech to use an open standard such as XML for their file format as the .ui files can then be repurposed for other uses such as XSL transformations to possibly link Qt Designer interfaces with web pages.

When the file is saved, a special tool called moc can be run on the file to convert it to the relevant C++ header and implementation file. Once the source code has been generated, you will have a boilerplate class for the interface available that can be used by re-implementing the class in your main code. You need to use polymorphism to use the class due to the fact that any modifications made to the generated class will be lost when you next generate the class. Using this method of re-implementing the class, you can then use the generated class and define your own slots of the same function name. This is a clever technique, and while a little confusing at first, gives the developer ultimate flexibility.

Qt Linguist

Qt Linguist is a tool for creating translations within Qt applications. The idea behind the tool is that you separate those who code the application from those who create translations. The traditional method of supporting multiple languages when developing software has been to implement multiple translations either within the code, or via a text file for each language. Qt prefers the more elegant technique of creating so called *translation files* that are created by the translators with Qt Linguist. Those files are then made use of by the developers in the Qt application. I like this technique because not only is it simple, but this kind of simplicity means that Qt developers can not only create multi-platform applications, but multi-platform, multi-language applications. I am sure that non-technical translators will appreciate this simplified method of dealing with translations, as well.

QMake

The final tool in the Qt toolbox (I will cover Qt Assistant in the next section) is QMake. This simple little tool lets you handle the building your applications easily. Many

developers spread their code out over multiple source files, and traditionally developers have needed to edit Makefiles, and other build scripts, to get their applications to build. When you roll in the multi-platform nature of Qt with different compilers and build environments, this could get real challenging real fast. QMake seeks to simplify the process, and when run will generate a make file for building your application. I have some experience with the GNU tools to do this, such as automake, and QMake is a welcome change. I found QMake not only makes the build system seamless, but due to the fact that it is bundled with Qt, makes it even easier.

Documentation and Qt Assistant

One of the major strengths of Qt is its incredible documentation. As a developer, documentation is always something that is important to have available due to the fact that every intimate detail of the API may need to be used, and as such should be well documented. Luckily, the Qt documentation team have done an incredible job at not only creating an impressive reference manual for every minute detail of the API, but have also included a number of other features in the documentation. These include:

Qt Community
Information about mailing lists, newsletters, bug reporting and more.
Getting started
Details on how to begin programming with Qt. This section also includes two full tutorials, and many examples.
API Reference
A full and complete reference of every class, function, and definition in the Qt toolkit. The reference is concise, and easy to read.
Modules
Documentation, tutorials, and examples of each of the additional modules within Qt.
Overviews
This section provides a number of walkthrough, and discussion documents on various parts of the Qt toolkit, such as the Qt object modal, and signals and slots.
Porting and platforms
This section gives information, and details on supporting each of the different platforms for your project, compilation details, and specific platform notes.
Tools
This section provides documentation for each of the tools included with Qt, such as Qt Designer, Qt Linguist, Qt Assistant, and QMake.
Licenses and credits
This section provides information about the different licenses available with Qt.

The documentation available with Qt is in HTML format, and can be viewed in any web browser (as well as being available at http://doc.trolltech.com). Although a web browser suffices for basic viewing of the documentation, Qt includes a special tool for dealing with the documentation called Qt Assistant. Qt Assistant provides an interface for searching, and querying the documentation, and viewing it. Although a simple front-end to the documentation, Qt Assistant provides a more integrated method of reading the class reference/information instead of a web browser. I am impressed that Trolltech takes documentation this seriously, and have committed to not only providing good documentation, but a tool to access it.

In use

Qt is a powerful toolkit, and has been carefully developed. When using the toolkit, you always get a true feeling of quality with what you are doing. This is largely from the tried and tested development model that has been created by Trolltech. Using C++ for graphical development is a natural choice due to its OOP philosophy, and it is very rare that you are in a position where you can see no elegant way of doing something. The available number of classes in Qt is impressive, and there is a convenience class for most common processes involved with modern software development. Not only do the classes allow you to create functionality easily, but each class has a rich API, consisting of a number of methods that are useful in most situations.

Qt Designer is an integral part of the Qt system, and generally works well for most applications. A few releases back Qt Designer was considerably more primitive than it is these days, and the latest version of Qt Designer is quite a mature and useful tool. While not a fully RAD tool (in the sense that it does not actively generate code that is embedded directly in your project), it is about as RAD as you want it to be – it generates the code, and *you* put it into your project. This prevents the kinds of spaghetti code problems that are often associated with RAD tools.

Although it is possible to review and evaluate Qt in a sterile environment where only Qt is used, in the real world Qt faces stiff competition from the likes of Java, Cocoa, GTK, and others. From my experience, Qt stands up well to these competitors, and offers a compelling, and in many cases, superior product. The main areas that many developers seem to look at when evaluating a product is its ease of use, functionality, and stability. From my testing, Qt seems to stand up on all of these points and I cannot really think of many areas in which Qt would be unsuitable for development. One testament to the nature of Qt, in my opinion, is the extent to which it is used. Because of the dual licensing of the toolkit, and the availability of a GPL version, Qt has been used by the UNIX based KDE project (www.kde.org). The sheer scope of that project is a good demonstration of the ability of Qt as a toolkit. KDE can be installed on Mac OS X using the Fink system (fink.sourceforge.net).

New in 3.2.2

Qt 3.2 has implemented a number of new features, in addition to the already impressive set of the 3.1 release. These new features include:

New Splashscreen class
This new class provides a contemporary splashscreen class that can be used to show a splash screen while your program is loading. The class will also allow you to display status messages in the splash screen.

New toolbox widget
A new widget has been added to create a toolbox that has collapsible areas. The widget was originally used in Qt Designer, and not available as a common class. Now it is, and can be added using the Qt Designer interface.

Improved menu editor in Qt Designer
The menu editor in Qt Designer has always worked fine, but not worked as intuitively as you might have expected. It has been revamped in 3.2.2.

Thread local storage
Multi-threaded applications can be written in Qt, and a new class has been added to store thread variables between threads.

Input masks
Input masks for controlling what input is added to a text entry box has been added. This makes it easier to validate user input.

Improved SQL support
With the addition of a DB2 driver, there has been a revamp of the SQL support in Qt, and data aware widgets work better with queries now.

Improved indic and syriac support
Support for right to left languages has been improved, with full support for indic and syriac languages.

Improved printer setup dialog
Additional functionality has been added to the printer setup dialog.

There was also a long list of user and developer submitted bugs that were fixed with this release, and various changes to particular platform versions of Qt.

Conclusion

Qt is a great toolkit. I am impressed with the technical structure of the software, the feature set, and the documentation that is included. Qt seems to offer a nice combination of hard core power, and RAD development. C++ is a good choice for a language to base the toolkit on, although it would be nice to see bindings for Java, or possibly even PHP. The toolkit not only provides a good range of graphical components, but also provides high performance data handling classes, and nice additions, such as data aware database widgets and classes, OpenGL support, and other features. The addition of Qt Assistant, Qt Linguist, and particularly Qt Designer, are sensible choices by Trolltech. Qt Designer, in particular, rapidly increases development speed.

If you are looking for a toolkit where you can write your code once, and run it on a number of different platforms, Qt is well worth looking in to. In these days of Windows, Linux, and Mac OS X, using a toolkit such as Qt makes sense. It makes particular sense for those developers creating free clients, and software to give away, as the GPL edition provides equivalent functionality to the commercial version, with only a change in licensing. The commercial version licensing allows you to distribute closed source applications commercially.

I look forward to seeing how Trolltech will expand Qt in the future, and look forward to seeing more Qt applications running on all my computers.

Peachpit

Essential books for the creative community

Upgrading to Panther?

Let Peachpit help you make an easy transition to Apple's newest operating system. Our wide selection of titles caters to every learning level and style so you're sure to find just the book to tame your Mac's inner beast.

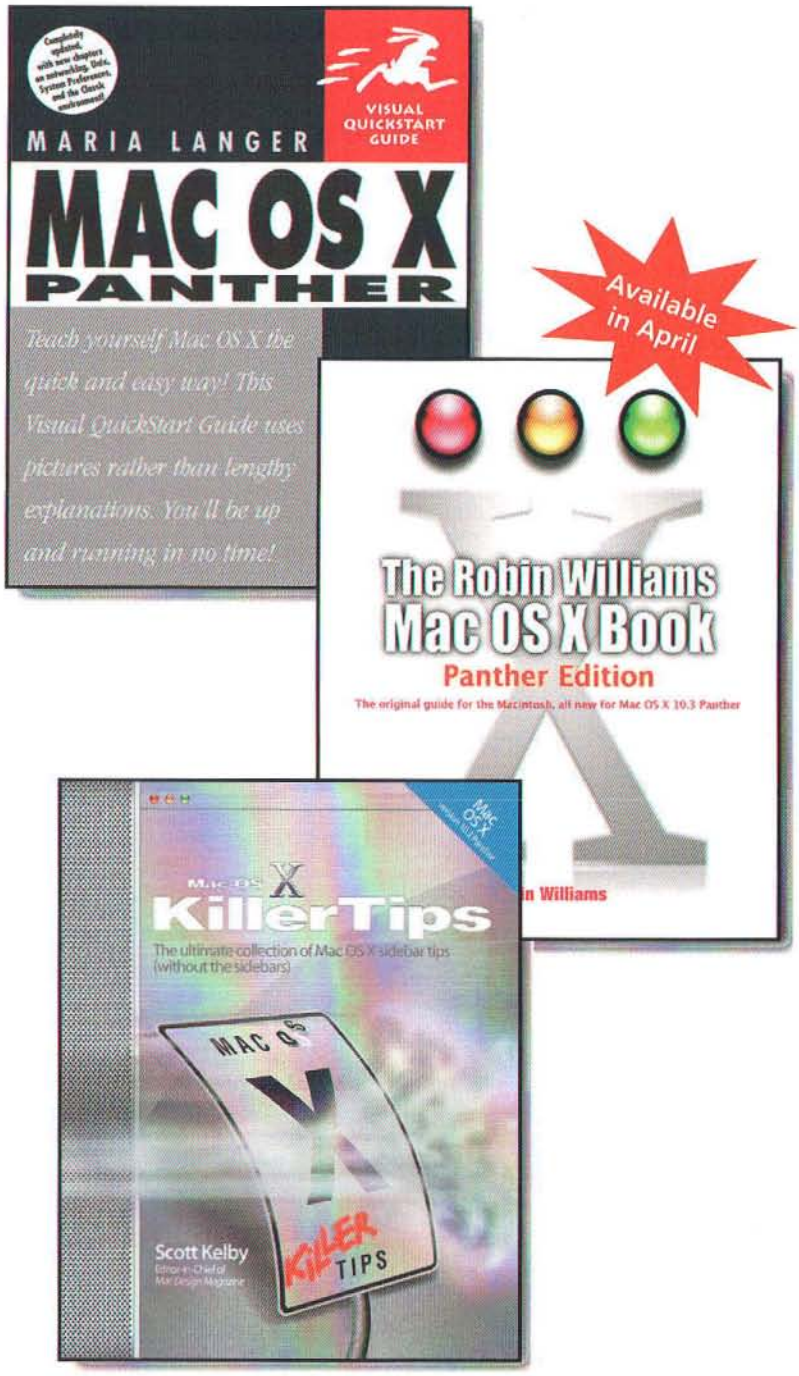
Mac OS X Conversion Kit 9 to 10 Side-by-Side, Panther Edition
By Scott Kelby
0-7357-1389-8 • $29.99

Mac OS X Panther Killer Tips
By Scott Kelby
0-7357-1393-6 • $29.99

Mac OS X 10.3 Panther: Visual QuickStart Guide
By Maria Langer
0-321-21351-3 • $24.99

Mac OS X Panther Hands-On Training
By Garrick Chow
0-321-24171-1 • $35.00

Mac OS X Security
By Bruce Potter, Preston Norvell and Brian Wotring
0-7357-1348-0 • $39.99



Coming in April!

Robin Williams Mac OS X Book, Panther Edition
By Robin Williams
0-321-23296-8 • $29.99

Save 30%!

Peachpit Press | New Riders

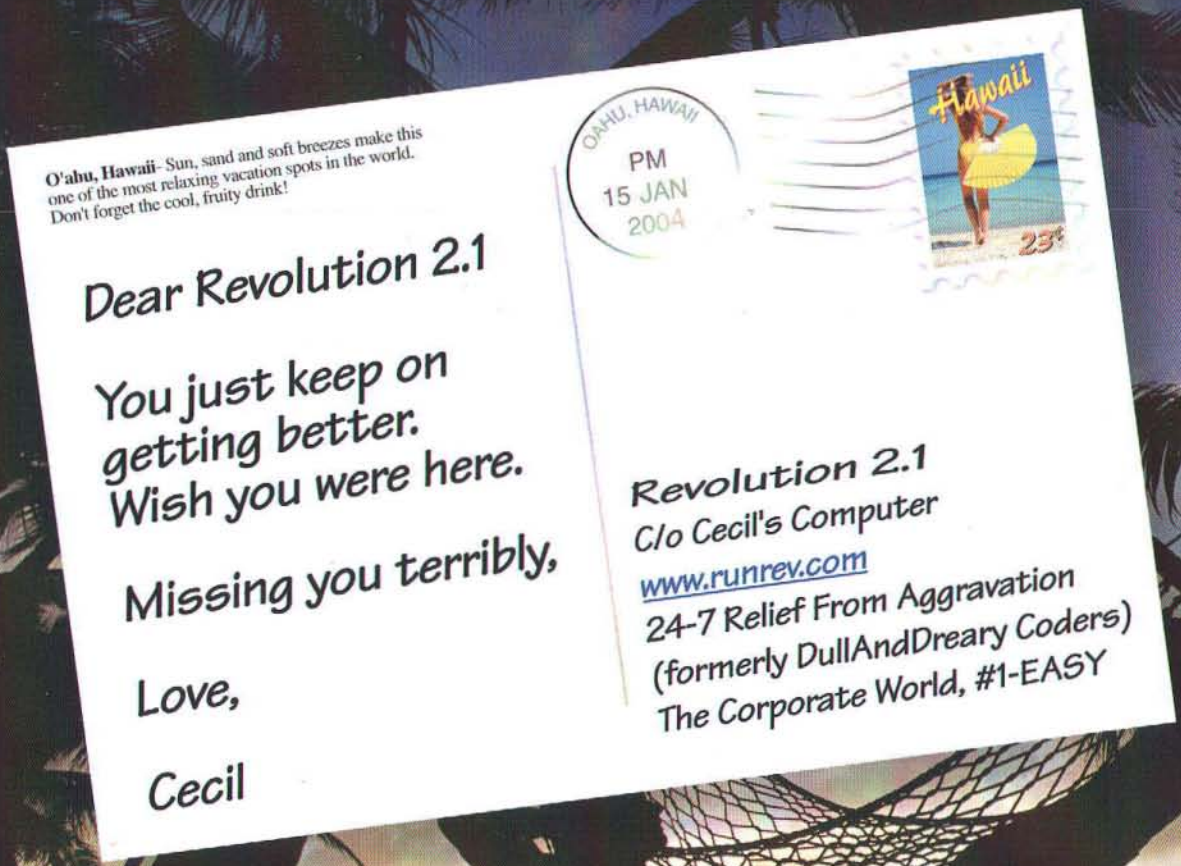
Buy these books today at www.peachpit.com/mactech0204 and save 30% off their retail price, plus enjoy free domestic U.S. shipping. Just enter coupon code EM-F4AA-MTMF when you get to our checkout page. It's that easy!

ADVERTISER/ PRODUCT LIST

List of Advertisers

List of Products

It'll Give You A Life.



But A Geek Is Always A Geek.

Revolution 2.1: the English like language designed around the way you think.

Develop and deliver on 14 platforms, including Mac OS X, Windows and Linux. Now with support for XML, additional SQL databases, video capture, Unicode, Reports, enhanced faceless CGI applications, and more.

And now from Dan Shafer, the first "how to" book on Revolution.
Get a headstart with "Revolution: Software at the Speed of Thought", volume one now shipping from www.runrev.com/revpress.
Thousands of developers have already joined the Revolution. Can you afford to wait?
Pricing starts at $149. Don't let the revolution in coding start without you.

User Centric Development Tools

Revolution Studio

winner macworld eddys

RUNTIME REVOLUTION

Runtime Revolution • 91 Hanover Street • Edinburgh EH2 1DJ • UK
Phone +44 (0)131 718 4333 • Fax +44 (0) 845 4588487 • www.runrev.com • Email info@runrev.com